TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur Verteilte und Selbstorganisierende Rechnersysteme

Studienarbeit

Sicheres Verteilen von Konfigurationsdaten und Migrationsstrategie zum Trennen von Diensten und Datenbasis

Sebastian Wehrmann

Chemnitz, den 31. Juli 2006

Betreuer: Dr. Kurt Tutschku
Dr. Jörg Anders

Aufgabenstellung

Aus historischen Gründen war die CSN Datenbank und die darauf zugreifenden Dienste immer auf demselben Rechner. Zum einen aus Geldmangel, zum anderen, weil die Verteilung der Konfiguration und Zugriffssteuerung zur Datenbank ein ungelöstes Problem ist. Aufgabe dieser Arbeit ist die physikalische und logische Trennung der Firewall (und des Shapers) von der Datenbank. Dazu muss ein Dienst geschaffen werden, der die Konfigurationsinformationen für die Firewall und potentiell andere Applikationen bereitstellt. Der Zugriff auf diese Informationen muss vor Dritten geschützt werden. Im Weiteren soll eine Migrationstrategie entworfen werden, wie der Übergang zu der skizzierten Lösung bewerkstelligt werden kann.
Erwartet wird eine Testimplementierung unter Beachtung des Radius-Servers für den Zugang der WLAN-Klienten. Ferner eine Dokumentation, die die Einarbeitung durch Programmierer ermöglicht, welche die Arbeiten fortsetzen wollen.

Hinweise:

- Denkbar wäre, die Konfigurationsfiles auf einem Webserver zu hinterlegen und von dort abzuholen (pull)
- Es ist aber auch gewünscht, Änderungen aktiv propagieren zu können (push).
- Dazu sind geignete Möglichkeiten zu untersuchen (finger?)

Eckdaten CSN-Server:

- Dual Xeon 2,4 GHz (4GB RAM)
- Betriebssystem: Linux (Debian Sarge 3.1)
- Datenbank: PostgreSQL 7.4
- Webserver: Apache 1.3
- DNS: Bind9

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

Accounting Erfassung des Datenvolumens

ASCII American Standard Code for Information Interchange

BNC Bayonet Nut Connector

CL Case Latency

CPU Central Processing Unit

CSA Communication Streaming Architecture

CSN Chemnitzer StudentenNetz

DB Datenbank

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

FAQ Frequently Asked Questions

GB Gigabyte

Gbit/s Gigabit pro Sekunde

HTTP Hypertext Transfer Protocol

MAC Media Access Control

MB Megabyte

Mbit/s Megabit pro Sekunde

IP Internet Protocol

LED Light Emitting Diode

RAM Random Access Memory

RPC Remote Procedure Call

SCP Secure Copy Protocol

SSH Secure Shell

SMP Symmetric Multiprocessing

URL Uniform Resource Locator

WWW World Wide Web

XML Extensible Markup Language

1 Einleitung

1.1 Das Chemnitzer StudentenNetz

Seit 1994 ist es Studenten im Wohnheim der TU Chemnitz gestattet, ihre Rechner an das Campusnetz anzuschließen. Dabei können die Studenten auf aktuelle Netzwerktechnik zugreifen. So sind mittlerweile alle Wohnheime mit einer Geschwindigkeit von 10 bzw. 100 Megabit pro Sekunde (Mbit/s) über Twisted Pair angebunden. Die alte Bayonet Nut Connector (BNC)-Technik wurde Mitte 2006 auch aus dem letzten Wohnheim verbannt. Neun Wohnheime sind ans Internet angeschlossen. Zwischen den Wohnheimen liegt Glasfasertechnik. Ausnahmen: Thüringer Weg 3 (Laser-Link) und Reichenhainer Str. 51 (Light Emitting Diode (LED)-Link). Eine Anbindung dieser beiden Wohnheime über Glasfaser ist aufgrund der (insbesondere Wetter-) Anfälligkeit des LED-/Laser-Link für 2007 geplant. Eine grafische Veranschaulichung der Netztopologie im Chemnitzer StudentenNetz (CSN) bietet Abbildung 1.1.

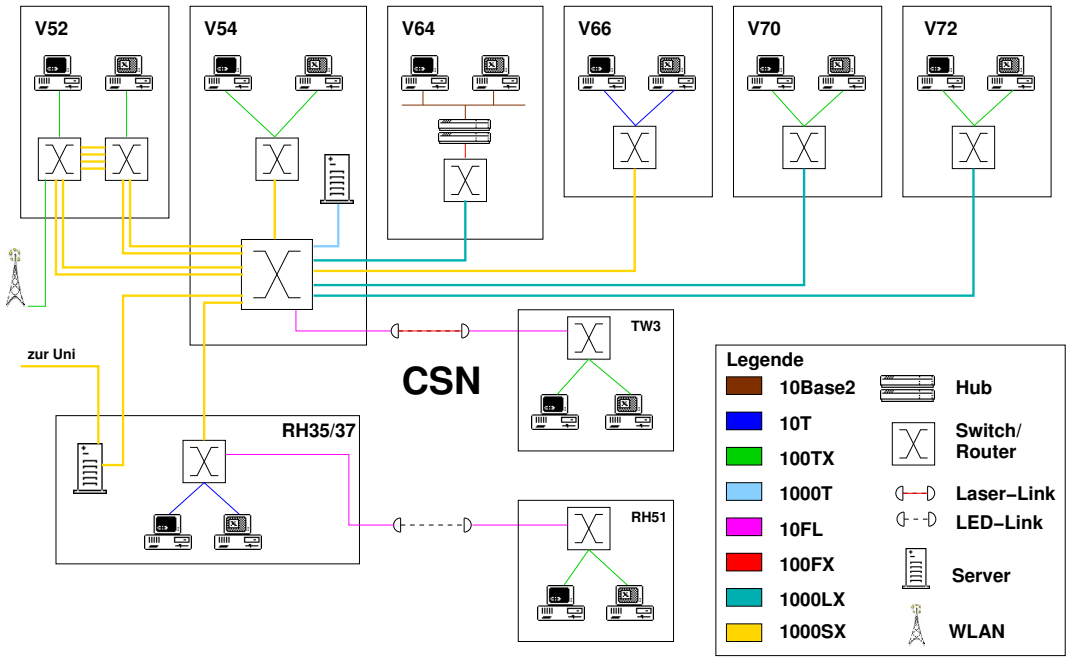

Abbildung 1.1: Netzstruktur des CSN (Stand April 2006)

Aktuell sind über 1700 Nutzer im CSN angeschlossen. Der CSN-Server ist trotz seiner enormen Kapazität dauerhaft unter Volllast und erweist sich immer mehr als Flaschenhals. Da die Fülle an Aufgaben, die der Rechner täglich zu erledigen hat, stetig wächst, ist seine Leistungsgrenze bald erreicht.
Mit dieser Studienarbeit soll Diskussionsmaterial über die Lastverteilung im CSN zur Verfügung gestellt werden, um den anstehenden Engpass möglichst kostengünstig und langfristig zu beheben.

1.2 Dienste auf dem CSN-Server

Ein großes Netzwerk wie das CSN benötigt zur automatischen Verwaltung eine Vielzahl an Diensten. Diese werden zum Großteil auf dem CSN-Server ausgeführt. Die Wichtigsten sollen im Folgenden kurz erläutert werden.

- **PostgreSQL 7.4**
  Die Datenbank enthält sämtliche Nutzerinformationen, Konfigurationseinstellungen und Trafficstatistiken. Sie muss ausserdem für die meisten Verwaltungsscripte, die CSN-Webseiten und einige Dienste erreichbar sein. Ausserdem werden mit Hilfe der Datenbank regelmäßig dynamische Konfigurationsdateien für die Dienste im CSN erstellt.

- **Apache 1.3**
  Der Apache-Webserver stellt die CSN-Homepage zur Verfügung. Über diese zum Großteil in Perl geschriebene Internetpräsenz können die Nutzer ihren Internetzugang verwalten, Hilfe bei Problemen suchen und sich über die Arbeit des CSN informieren. Ausserdem gibt es für Mitarbeiter einen internen Bereich, der detaillierte Informationen über die eingesetzten Techniken bereitstellt.

- **DHCP**
  Jedem CSN-Nutzer wird nach erfolgreicher Anmeldung eine feste Internet Protocol (IP)-Adresse zugeteilt. Diese IP-Adresse steht, gemeinsam mit der Media Access Control (MAC)-Adresse der Netzwerkkarte des Nutzers, in der Datenbank. Der Dynamic Host Configuration Protocol (DHCP)-Server teilt mit Hilfe dieser Informationen dem Nutzer-Rechner seine IP-Adresse samt Netzmaske, die Gateway-Adresse und die Adresse des Domain Name Service (DNS)-Servers über DHCP mit, sodass ein manuelles Eintragen dieser Informationen nicht nötig ist. Zusätzlich sind die Switches so konfiguriert, dass die Nutzer nur mit Ihrer IP/MAC-Kombination ins Netzwerk gelangen können.

- **weitere Dienste**
  Des Weiteres läuft auf dem CSN-Server noch eine Firewall (iptables), ein DNS-Server, ein Mailserver (exim) für die CSN-Mailadressen und diverse Scripte z.B. zum Erzeugen der Konfigurationsdateien, zum Neustart der Dienste oder Konfigurieren der Netztechnik.

Diese Dienste in ihrer Vielzahl verursachen auf dem CSN-Server eine große Last. Im Rahmen dieser Studienarbeit soll zur Lastverteilung die Datenbank auf einen zweiten Rechner (Datenbank (DB)-Server) ausgelagert werden. Dadurch sind Wege aufzuzeigen, wie die anderen Dienste weiterhin ihre Konfigurationsdateien bekommen.

1.3 anzuschaffende Hardware

Um die Datenbasis vom CSN-Server zu trennen, ist ein zweiter Computer nötig, welcher die Datenbank aufnimmt. Er wird damit zum DB-Server und muß folgende Kriterien erfüllen:

- **Geschwindigkeit:**
  Aufgrund der Vielzahl an Anfragen, muss der Datenbankserver in angemessen kurzer Zeit reagieren, die Anfragen bearbeiten und beantworten.
- **Ausfallsicherheit:**
  Ein Ausfall des Datenbankservers bedeutet den Ausfall sämtlicher Dienste des CSN. Die meisten Perl-Scripte, die ein stabiles und sicheres Netz gewährleisten, greifen auf die Datenbank zurück, um Informationen oder Einstellungen zu verarbeiten.
- **Kosten:**
  Auch das CSN muss seine Ausgaben rechtfertigen. Insofern muss darauf geachtet werden, welche preislichen Alternativen es beim Hardware-Kauf gibt, und ob ein vermeintlich günstigeres Angebot nicht durch zusätzliche Defekte und Wartungsarbeiten den Preisvorteil zunichte machen könnte.

In der PostgreSQL-Frequently Asked Questions (FAQ) [5] wird dieser wichtige Punkt kurz beantwortet: PostgreSQL läuft, frei übersetzt, auf fast jeder Hardware, aber wenn Ausfallsicherheit und Performance wichtig sind, sollte auf qualitativ hochwertige Hardware zurückgegriffen werden.
Wichtig für eine gute Performance ist der Arbeitsspeicher (Random Access Memory (RAM)). Dieser sollte nicht nur groß, sondern auch schnell sein. Da die CSN-Datenbank im Moment eine Größe von etwa 5 bis 6 Gigabyte (GB) hat, ist ein Arbeitsspeicher von 2 GB für die Datenbank in meinen Augen ausreichend. Berechnet man den für das Betriebssystem anfallenden RAM noch dazu, sollte der DB-Server mit 2 bis 4 GB bestückt werden. Anbieten würden sich hier also zwei oder vier 1024 Megabyte (MB) Riegel, die mit mind. 400 Mhz, besser 533 Mhz, getaktet sind und einen möglichst niedrigen Case Latency (CL)-Wert haben, da ein niedriger CL-Wert schnellere Lese- und Schreibzugriffe garantiert.
Die Central Processing Unit (CPU) sollte aufgrund der vielen (auch parallelen) Anfragen eine Dual-Core-CPU bzw. ein Symmetric Multiprocessing (SMP)-System sein. Um für die Zukunft gerüstet zu sein, empfiehlt es sich wegen der größeren Verarbeitungseinheit 64 Bit-CPUs zu kaufen.

Da die anderen Dienste nicht annähernd so viele Ressourcen benötigen wie die Datenbank, bietet es sich an, den aktuellen CSN-Server als neuen Datenbank-Server zu benutzen und einen neuen PC für die restlichen Dienste zu kaufen. Dieser wird zum neuen CSN-Server.
Intel hat eine Architektur entwickelt, ideal für PCs, die viel Netzwerktraffic zu verarbeiten haben (z.B. Firewall bzw. Netzwerkrouter). Die Communication Streaming Architecture (CSA) verbindet den Memory-Controller direkt mit der Netzwerkkarte und verhindert somit eine Überlastung des PCI-Bus und steigert die Verarbeitungsgeschwindigkeit erheblich. Mehr Informationen hierzu sind dem Whitepaper [4] zu entnehmen. Ausserdem sollte aufgrund des zusätzlichen Overhead von dem Kauf eines Mehrprozessorsystems Abstand genommen werden.
Schlussfolgernd sollte eine schnelle Intel-CPU mit 2 GB RAM für den CSN-Server gekauft werden.

1.4 Standortbestimmung

Da das CSN über zwei Serverräume im Wohnheim Vetterstrasse 54 bzw. Reichenhainer Str. 35 verfügt, sollten die beiden Server getrennt aufgestellt werden. Damit wird vermieden, dass beide Rechner bei Wasserschaden, Brand, Blitzeinschlag o.ä. gemeinsam ausfallen.
Aufstellungsvorschlag:

- **CSN-Server**
  Reichenhainer Str. 35
- **DB-Server**
  Vetterstraße 54

Der DB-Server wird **db1**.csn.tu-chemnitz.de, der CSN-Server **csn-server**.csn.tu-chemnitz.de benannt, auch weil in Zukunft eine Replikation der Datenbank angedacht ist (siehe Kapitel 4 Ausblick).
Abbildung 1.2 zeigt die CSN-Netzstruktur nach der Migration.

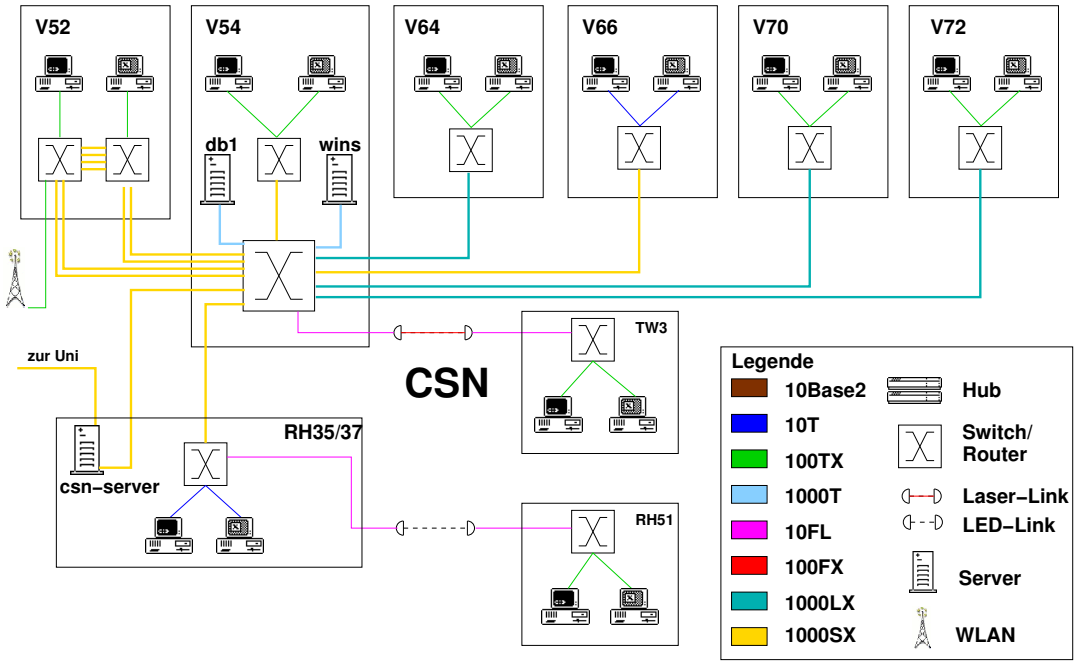

Abbildung 1.2: Netzstruktur des CSN nach dem Umbau

2 Lastverteilung im CSN

2.1 Vorüberlegung

In diesem Kapitel sollen nun Varianten besprochen werden, wie die Konfigurationsdateien zwischen Diensten und Datenbasis aufgeteilt werden können. Dabei bezeichne ich den Server, der die Datenbasis vorhält und die Konfigurationsdateien erzeugt, als DB-Server; den Server, auf dem die Dienste laufen, als CSN-Server.

2.2 Variante 1: Apache, wget, cfingerd/ssh

Die erste und einfachste Variante zum Verteilen der Konfigurationsdateien ist eine Kombination aus Webserver (`Apache`) und einem Tool zum Anfordern von Dateien über Hypertext Transfer Protocol (HTTP) (`wget`). Die Konfigurationsdateien werden in einem zentralen, fest vorgegebenen Verzeichnis auf dem DB-Server abgelegt und vom CSN-Server per `wget` in regelmäßigen Abständen geholt.

2.2.1 Konfiguration des Apache

Das zentrale Konfigurationsverzeichnis ist gegen unbefugten Zugriff Dritter zu schützen. Dies kann zum Beispiel über eine Access-List erfolgen. Folgende Optionen sollten in der `httpd.conf` aktiviert werden:

- `Options -Indexes`
  Deaktivieren der Verzeichnislisten
- ```
  Deny from all
  Allow from 134.109.101.249
  Order allow,deny
  ```
  Zugriff explizit nur vom CSN-Server (134.109.101.249)

Desweiteren bietet `Apache` eine Möglichkeit, Benutzer zu authentifizieren. So muss ein fester Benutzer mit einem fest zugewiesenen Passwort erstellt werden. Das Passwort muss auf beiden Rechner bekannt sein und sollte schwer zu erraten sein. Ich verwende hier als Beispiel den Benutzer `config` und das Passwort `passwort`, erwarte aber, dass in der späteren Umsetzung ein sicheres Passwort (mind.
  ```

10 Stellen bestehend aus Buchstaben, Zahlen und Sonderzeichen) verwendet wird. Das Verzeichnis wird dann über folgende Befehle geschützt:

- `AuthType Basic`
  Art der Authentifizierung
- `Require User config`
  Erlaubt nur dem User `config` den Zugriff
- `AuthUserFile /etc/apache/passwd/config.htpasswd`
  Pfad zur Passwortdatei

Das Passwort des Benutzers `config` wird unter `/etc/apache2/passwd/config.htpasswd` gespeichert. Der Inhalt dieser Datei setzt sich aus einem Benutzer- und einem Passwort-Feld zusammen, wobei die Passwörter mit der Funktion `crypt()` verschlüsselt werden. So würde der Inhalt der Datei für den Benutzer `config` und das Passwort `passwort` wie folgt aussehen:

- `config:nleFKWv6rFWOI`
  Kombination aus Benutzername:Passwort (verschlüsselt)

Ich weise nochmals darauf hin, dass während der Implementierung ein sicheres Passwort zu verwenden ist!
Listing 2.1 zeigt abschließend nochmal den Auszug aus der `httpd.conf` zusammengefasst.

```
1 <Directory "/var/www/configs">
2   Options -Indexes
3   Deny from all
4   Allow from 134.109.101.249
5   Order allow,deny
6   AuthType Basic
7   Require User config
8   AuthUserFile /etc/apache/passwd/config.htpasswd
9 </Directory>
```

Listing 2.1: Auszug aus der Datei httpd.conf

Für den Zugriffsschutz und die Authentifizierung sind die Module `mod_access` und `mod_auth` in der `httpd.conf` mittels `LoadModule` zu laden.
Da einige Dienste (z.B. die Firewall) mehrere Konfigurationsdateien benötigen, ist es empfehlenswert, die Dateien in ein Archiv zu legen, und nur dieses zu holen. Dazu muss die Datei `userconfig_create`, welche die Konfigurationsdateien erstellt, insoweit erweitert werden, dass

nach dem Erzeugen der Dateien diese gepackt und das Archiv auf den Webserver geladen wird. Listing 2.2 zeigt die nötigen Änderungen für Firewall und DNS. Hierbei wird (siehe Zeile 8,9 bzw. 18,19) mit Hilfe der Funktion `qx(Befehl);` der Befehl zum packen der Konfigurationsdateien (`tar cfv archiv.tar /pfad/zu/dateien`) und zum Verschieben in das zentrale Konfigurationsverzeichnis auf dem Webserver (`mv archiv.tar /var/www/configs`) auf die Konsole geschickt und dort ausgeführt.

```
1  ...
2  #
3  # Firewall config erzeugen
4  # 09/2003 by hje
5  #
6  sub fwconfig() {
7     ...
8     qx(tar cvf configs_iptables.tar $FWBASE/config);
9     qx(mv configs_iptables.tar /var/www/configs/);
10 }
11 ...
12 #
13 # Zonefile Generierung fuer den Bind
14 # 2005 by marks
15 #
16 sub dnsconfig () {
17    ...
18    qx(tar cvf configs_bind.tar $ZONEBASE);
19    qx(mv configs_bind.tar /var/www/configs/);
20 }
21 ...
```

Listing 2.2: Änderungen an userconfig_create

2.2.2 Beziehen der Konfigurationsdateien

`Wget` ist ein Konsolen-Tool für den schnellen und flexiblen Download von Dateien aus dem Web. Sollte es auf dem CSN-Server nicht installiert sein, ist dies als `root` nachzuholen mittels `apt-get install wget`.
Für den gesamten Vorgang muss ein neuer Systembenutzer angelegt werden, welcher die Konfigurationsdateien holt, sie auf die Dienste verteilt und diese anschließend neu startet. Er benötigt also

Schreibrechte in den Konfigurationsordnern (setzbar über `chown` und `chmod`) und root-Rechte für den Dienste-Neustart. Listing 2.3 zeigt den Befehl zum Anlegen des Benutzers. Hier wird der Systembenutzer `cfggrabber` mit dem Homeverzeichnis `/home/cfggrabber` und deaktiviertem Passwort (kein Login möglich) erstellt.
Die Rechtevergabe zum Neustart der Dienste geschieht über die Datei `/etc/sudoers`. Diese muss als `root` über das Tool `visudo` bearbeitet werden. Der Eintrag für den cfggrabber ist Listing 2.4 zu entnehmen. Der Neustart der Dienste wird dann über das Script `userconfig_execute` durchgeführt, welches unter `/var/CSN/bin/` zu finden ist.

```
1 adduser --system --home /home/cfggrabber --shell /bin/bash \
2 --disabled-password cfggrabber
```

Listing 2.3: Anlegen des Systembenutzers cfggrabber

```
1 cfggrabber        ALL = /etc/init.d/dhcp, /etc/init.d/bind
```

Listing 2.4: Rechtevergabe mittels visudo

Wget kann mit Hilfe einiger Parameter konfiguriert werden. Die folgenden Optionen sind wichtig:

- `--http-user`
  setzt den Benutzernamen, mit welchem sich am `Apache` authentifiziert werden soll
- `--http-password`
  setzt das Passwort, mit dem sich am `Apache` authentifiziert werden soll
- `--no-cache`
  weist den Server an, keine gecacheten, sondern die neuesten Dateien zu schicken (`Pragma: no-cache` im HTTP-Header)

Ein Beispielaufruf zum Holen der DynShaper-Konfiguration ist Listing 2.5 zu entnehmen. Zusätzlich zu den oben erklärten Parametern wird noch die Uniform Resource Locator (URL) zur Konfig-Datei übergeben.

```
1 wget --http-user config --http-password passwort --no-cache \
2       http://db1.csn.tu-chemnitz.de/config/dynshaper.conf
```

Listing 2.5: Beispielaufruf wget

`wget` bietet zur Verbesserung der Usability die Möglichkeit, Einstellungen in einer zentralen Konfigurationsdatei zu speichern. Diese muss ins Homeverzeichnis des aufrufenden Benutzers unter dem Namen `.wgetrc` gespeichert werden. Listing 2.6 zeigt den Inhalt dieser Datei. Hier kann zusätzlich ein Logfile eingestellt werden (Zeile 6), die Option `noclobber = on` in Zeile 7 verhindert, dass

`wget` die alten Konfigurationsdateien beim wiederholten Download sichert (z.B. konfig.1.bak, konfig.2.bak,...) und damit unsinnig Festplattenspeicher verbraucht.

```
cache = off
cookies = off
debug = off
http_user = config
http_password = passwort
logfile = /var/log/wget_config_log
noclobber = on
```

Listing 2.6: die Konfigurationsdatei .wgetrc

Damit `wget` die Dateien in regelmäßigen Abständen holen kann, muss es über cron, die Jobsteuerung von Unix, gestartet werden. Vorher sollten allerdings alle wget-Aufrufe, das Entpacken und Verschieben der Konfigurationsdateien sowie der Neustart der Dienste, in einer zentralen Bash-Datei (`/home/cfggrabber/cron_cfggrabber`, Listing 2.7) gespeichert werden.

```
!#/bin/bash
wget http://db1.csn.tu-chemnitz.de/config/configs_bind.tar
wget http://db1.csn.tu-chemnitz.de/config/dhcp.conf
wget http://db1.csn.tu-chemnitz.de/config/dynshaper.conf
wget http://db1.csn.tu-chemnitz.de/config/configs_iptables.tar
wget http://db1.csn.tu-chemnitz.de/config/macs
wget http://db1.csn.tu-chemnitz.de/config/static-arp
wget http://db1.csn.tu-chemnitz.de/config/static-arp.diffs
wget http://db1.csn.tu-chemnitz.de/config/valid_ips
cd /var/CSN/firewall/config && \
  tar -xfv /home/cfggrabber/configs_iptables.tar
cd /var/cache/bind/ && tar -xfv /home/cfggrabber/configs_bind.
    tar
mv /home/cfgggarbber/dhcpd.conf /etc/dhcpd.conf
mv /home/cfggrabber/dynshaper.conf /var/CSN/dynshaper/dynshaper
    .conf
chown shaper:root /var/CSN/dynshaper/dynshaper.conf
/var/CSN/bin/userconfig_execute
```

Listing 2.7: zentrale Bash-Datei cron_cfggrabber

Listing 2.8 zeigt den Eintrag in der crontab, der jede Viertelstunde die Konfigurationsdateien holt. Der Eintrag besteht aus 6 Spalten, die ersten 5 dienen der Zeitangabe (Minute, Stunde, Tag, Monat, Wochentag), die letzte Spalte enthält den auszuführenden Befehl.

```
1 */4 * * * * /home/cfggrabber/cron_cfggrabber
```

Listing 2.8: Cronjob für cron_cfggrabber

2.2.3 Propagieren von Änderungen

Angedacht ist eine Möglichkeit, Änderungen in den Konfigurationsdateien den anderen Rechnern mitzuteilen. Überlegt wurde eine Variante, in der der `Configurable finger Daemon (cfingerd)` zum Einsatz kommt. Dieser kann als Reaktion auf ein finger-Kommando lokal ein Script ausführen. So würde der `cfingerd` bei Eingang eines finger-Kommandos das Script `cron_cfggrabber` (siehe Listing 2.7) starten und somit ein Holen der Konfigurationsdateien anstoßen.
Allerdings hat diese Variante einige Nachteile. Der `cfingerd` besitzt keine Möglichkeit, Benutzer zu authentifizieren bzw. einen Missbrauch zu verhindern. Ein möglicher Angreifer könnte dem Rechner dauerhaft finger-Kommandos senden, um ein wiederholtes starten des `cron_cfggrabber`-Scripts zu erzwingen. Der dauerhafte Neustart der Dienste wäre fatal. Ausserdem ist das finger-Protokoll in meinen Augen nicht dafür gedacht, Scripte anzustoßen. Ursprünglich soll damit nur eine Möglichkeit geschaffen werden, Informationen über aktive Benutzer auf Unix-System zu erlangen.
Eine zweite Variante ist der Secure Shell (SSH)-Client. Dieser erlaubt es, sich auf dem Remote-System einzuloggen, einen Befehl zu starten und anschließend die Verbindung wieder zu trennen.
Der Login sollte idealerweise nicht mit einem Passwort erfolgen, da sonst ein automatisches Einloggen nicht möglich ist. Deshalb ist hier das Public-Key-Verfahren anzuraten, welches ohnehin mehr Sicherheit verspricht. Wie die Schlüssel für das Public-Key-Verfahren auf einem UNIX-System erstellt werden, zeigt Listing 2.9.

```
1 sweh@katzenklo:~$ ssh-keygen -t dsa
2 Generating public/private dsa key pair.
3 Enter file in which to save the key (/home/sweh/.ssh/id_dsa):
4 Enter passphrase (empty for no passphrase):
5 Your identification has been saved in /home/sweh/.ssh/id_dsa.
6 Your public key has been saved in /home/sweh/.ssh/id_dsa.pub.
7 The key fingerprint is:
8 ##:##:##:##:##:##:##:##:##:## sweh@katzenklo.local
9 sweh@katzenklo:~$
```

Listing 2.9: Erzeugen eines Schlüsselpaares

Der private Schlüssel muss auf den Rechner, der die Verbindung herstellt. Damit dieser nicht über das Internet verteilt werden muss, sollte das Schlüsselpaar auf dem DB-Server erstellt werden. Eine Passphrase zum Sichern des Schlüssel darf nicht eingegeben werden (Zeile 4), da sonst ein automatischer Login nicht möglich ist. Deshalb muss ein unberechtigter Zugriff auf den Schlüssel verhindert werden, indem nur der Benutzer (`cfggrabber`) Leserechte für den Schlüssel bekommt. Abschließend muss der öffentliche Schlüssel auf dem CSN-Server unter `.ssh/authorized_keys` gespeichert werden. Nun ist ein automatisches Anstoßen des wget nach Neuerstellen der Konfigurationsdateien mittels `ssh cfggrabber@csn-server.csn.tu-chemnitz.de cron_cfggrabber` möglich. Der Login erfolgt dank Public-Keys automatisch und ohne Passwortabfrage, im Anschluss wird auf dem Client-Rechner das `cron_cfggrabber`-Script gestartet und die ssh-Verbindung getrennt.

2.2.4 Vor- und Nachteile

Der initiale Arbeitsaufwand hält sich bei dieser Variante in Grenzen. Es müssen nur die Perl Scripte angepasst und die Programme konfiguriert werden. Auch sind die verwendeten Programme auf nahezu jedem Linux-System vorhanden und aufgrund ihrer starken Verbreitung nahezu Bug-frei. Nachteilig ist die große Starrheit des Systems. Es ist umständlich, neue Dienste mit Konfigurationsdateien hinzuzufügen. Auch werden zu viele unterschiedliche Programme benötigt, was bei Änderungen jener zu Kompatibilitätsproblemen führen kann. Hier wäre meiner Meinung nach eine zentrale Lösung besser.

Vorteile	**Nachteile**
genutzte Programme auf nahezu jedem System vorhanden	schwer erweiterbar
einfaches Einrichten des Systems	zu viele verschiedene Programme
	zu dezentral

Tabelle 2.1: Vor- und Nachteile der Variante Apache - wget - finger/ssh

2.3 Variante 2: Secure Copy Protocol

Variante 1 (siehe 2.2) bezog sich nur auf das Holen von Konfigurationsdateien. In dieser 2. Variante soll nun eine Möglichkeit vorgestellt werden, in der die Konfigurationsdateien gezielt versendet werden. Hierfür wird das Programm Secure Copy Protocol (SCP) zum sicheren Versenden der Dateien verwendet, sowie **ssh!** (ssh) zum Neustart der Dienste.

2.3.1 Versenden der Konfigurationsdateien

Um mittels SCP Dateien über das Netzwerk zu versenden, benötigt man auf Quell- und Zielrechner einen Account mit ssh-Zugang (siehe Listing 2.3). Zur Automatisierung bietet es sich an, wie in Li-

sting 2.9 den Zugang über das Public-Key-Verfahren zu realisieren. Es müssen auf dem DB-Server die Schlüssel erzeugt und der öffentliche Schlüssel auf den CSN-Server kopiert werden.
Nun ist ein automatischer Login möglich und Dateien können mittels
`scp <datei> cfggrabber@csn-server.csn.tu-chemnitz.de:<datei>`
über das Netzwerk geschickt werden.
SCP bietet die Möglichkeit, mehrere Dateien aus einem Ordner rekursiv zu versenden. Somit sind nur wenige Änderungen am Ursprungssystem nötig. Das Script `userconfig_create` muss so angepasst werden, dass es die Konfigurationsdateien in ein zentrales Verzeichnis (`/var/CSN/config`) mit entsprechenden Unterordnern speichert (Listing 2.10). Nun können mittels
`scp -r /var/CSN/config cfggrabber@csn-server.csn.tu-chemnitz.de:~/confinc/`
alle Konfigurationsdateien vom DB- zum CSN-Server geschickt werden. Auf dem CSN-Server ergibt sich dann ein genaues Abbild der Dateistruktur. Beim ersten Übertragen ist darauf zu achten, dass das Verzeichnis `/home/cfggrabber/confinc/` sowie die Unterverzeichnisse auf dem CSN-Server existieren, da diese nicht automatisch angelegt werden.
Ein automatisches Versenden der Dateien zu definierten Zeitpunkten ist mittels Cronjobs möglich. Einen Beispiel-Eintrag zeigt Listing 2.11.

```
1  ...
2  my $NEWFOLDER = "/var/CSN/config";
3  my $FWBASE = $NEWFOLDER."/var/CSN/firewall";
4  my $ZONEBASE = $NEWFOLDER."/var/cache/bind";
5  my $DHCPDCONF = $NEWFOLDER."/etc/dhcpd.conf";
6  my $ARPCONF = $NEWFOLDER."/var/CSN/tftpd/static-arp";
7  my $ARP_DIFFS = $NEWFOLDER."/var/CSN/tftpd/static-arp.diffs";
8  my $DSCONF = $NEWFOLDER."/var/CSN/dynshaper/dynshaper.conf";
9  my $MACCONF = $NEWFOLDER."/var/CSN/tftpd/macs";
10 my $ACCOUNTINGCONF = $NEWFOLDER."/var/CSN/accounting/valid_ips"
       ;
11 my $FORWARDCONF = $NEWFOLDER."/db.csn.tu-chemnitz.de";
12 my $DNSCHANGE = $NEWFOLDER."/var/run/bind/changed";
13 ...
```

Listing 2.10: Änderungen an der Datei userconfig_create

```
1  */4 * * * * scp -r /var/CSN/config cfggrabber@csn-server.csn.tu
       -chemnitz.de:~/confinc/
```

Listing 2.11: automatisches Holen der Konfigurationsdateien mittels cron

2.3.2 Neustart der Dienste

Nach dem Versenden der Konfigurationsdateien liegen diese im home-Verzeichnis des Benutzers `cfggrabber`. Benötigt wird nun ein Script, welches die Konfigurationsdateien auf die Dienste verteilt, und diese anschließend zu einem Reload bzw. Neustart bringt (Listing 2.12). Das Verteilen der Konfigurationsdateien geht aufgrund des eindeutigen Abbildes der Verzeichnisstruktur im Ordner `/home/cfggrabber/confinc` sehr einfach (Zeile 2). Der Neustart der Dienste geschieht über die Datei `/var/CSN/bin/userconfig_execute` (siehe Zeile 3).

```
1 !#/bin/bash
2 cd /home/cfggrabber/confinc/ && cp -r * /
3 /var/CSN/bin/userconfig_execute
```

Listing 2.12: Bash-Datei cfg_restart zum Neustart der Dienste

Anschließend muss die Datei `cfg_restart` nach `/var/CSN/bin/` verschoben und mittels `chmod u+x` ausführbar gemacht machen. Nun kann mittels

```
ssh cfggrabber@csn-server.csn.tu-chemnitz.de cfg_restart
```

das Script auf dem CSN-Server ausgeführt und die Dienste neugestartet werden.

2.3.3 Vor- und Nachteile

Die 2.Variante kommt mit viel weniger Konfigurationsaufwand aus als die 1. Ausserdem wird ein geringerer Pool an Programmen benötigt, somit ist auch der Wartungsaufwand geringer. Nachteilig bleibt noch zu erwähnen, dass auf beiden PCs ein SSH-Login erforderlich ist. Eine Lösung ohne diesen wäre vorteilhafter.

Vorteile	**Nachteile**
geringer Konfigurationsaufwand	SSH-Login und System-User erforderlich
wenige Programme nötig	
einfache Installation	

Tabelle 2.2: Vor- und Nachteile Variante SCP - SSH

2.4 Variante 3: XML-Remote Procedure Call

2.4.1 die Spezifikation von XML-Remote Procedure Call (RPC)

Mit Remote Procedure Calls (RPC) [6] können auf einem Server definierte Funktionen (procedures) über das Netzwerk (remote) aufgerufen (call) werden. Das Prinzip beruht dabei auf dem Client-Server-Prinzip: der Server hat einen definierten, endlichen Pool an Funktionen, die er den Clients anbietet. Die

Clients rufen über das Netzwerk eine dieser Funktionen auf, und erhalten vom Server das Ergebnis. Die allgmeine Funktionsweise von RPC ist in Abbildung 2.1 schematisch dargestellt.

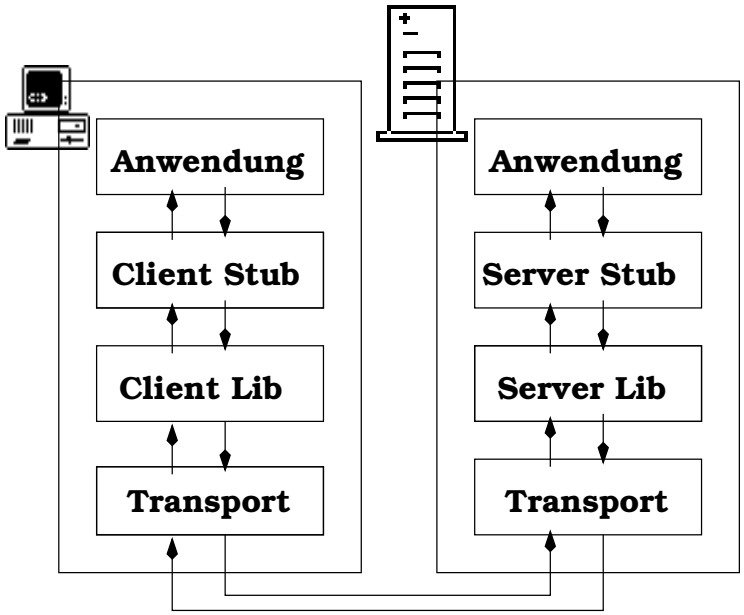

Abbildung 2.1: Schema der Funktionsweise von RPC

Die Extensible Markup Language (XML) ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur, der vom World Wide Web Consortium (W3C) definiert wird. [1] Die zwischen Client und Server versendeten Daten werden in XML-Markup verpackt. Eine Beispielanfrage des Client zeigt Listing 2.13. Die Zeilen 1 bis 5 enthalten den HTTP Header, welcher mindestens den User-Agent und Host sowie Content-Type (immer `text/xml`) und Content-length enthalten muss. Zeile 7 bis 15 enthalten dann die eigentlichen Daten im XML-Format. Dabei wird die Methode `getStateName` angefragt, und der Parameter `41` (Name des Bundesstaates 41) übergeben. Der Server bekommt diese Daten, decodiert das XML-File und ruft lokal die Funktion `getStateName` auf. Das Ergebnis (South Dakota) schickt er an den Client zurück (Listing 2.14). Im Falle eines Fehlers im Server wird eine Fehlermeldung generiert (`<fault>`) und an den Client gesendet, welcher diesen Fall extra abfragen und entsprechend reagieren muss. Listing 2.15 zeigt beispielhaft eine solche Fault-Response des Servers.

```
1 POST /RPC2 HTTP/1.0
2 User-Agent: Frontier/5.1.2 (WinNT)
3 Host: betty.userland.com
4 Content-Type: text/xml
5 Content-length: 181
6
```

```
<?xml version="1.0"?>
  <methodCall>
    <methodName>examples.getStateName</methodName>
    <params>
      <param>
        <value><i4>41</i4></value>
      </param>
    </params>
  </methodCall>
```

Listing 2.13: Beispiel eines XML-RPC-Request

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
  <methodResponse>
    <params>
      <param>
        <value><string>South Dakota</string></value>
      </param>
    </params>
  </methodResponse>
```

Listing 2.14: Beispiel einer XML-RPC-Response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
  <methodResponse>
```

```
10      <fault>
11        <value>
12          <struct>
13            <member>
14              <name>faultCode</name>
15              <value><int>4</int></value>
16            </member>
17            <member>
18              <name>faultString</name>
19              <value><string>Too many parameters.</string></value>
20            </member>
21          </struct>
22        </value>
23      </fault>
24    </methodResponse>
```

Listing 2.15: Beispiel eines XML-RPC-Fault

XML-RPC unterstützt die folgenden einfachen Datentypen:

Tag	**Typ**	**Beispiel**
<i4> oder <int>	Vier-Byte vorzeichenbehafteter Integer	-12
<boolean>	Wahrheitswerte (0 oder 1)	1
<string>	Zeichenkette (string)	Hallo Welt
<double>	Fließkommazahl (vorzeichenbehaftet)	-12.214
<dateTime.iso8601>	Datum/Zeit	19980717T14:08:55
<base64>	Binärdaten	eW91IGNhbgdGhpcyE=

Tabelle 2.3: Datentyen in XML-RPC

Zusätzlich zu diesen einfachen Datentypen gibt es noch die `<array>` und `<struct>` Datentypen. Ein `<struct>` enthält `<member>` und jedes `<member>` enthält einen `<name>` und eine `<value>`. Der `<struct>` entspricht also einem assoziativen Array. Ein `<array>` besteht aus einem einzelnen `<data>`-Feld, welches mehrere `<value>` enthalten kann. Sowohl `<struct>` als auch `<array>` sind rekursiv, das heisst, ein `<array>` kann ein oder mehrere `<struct>` enthalten und umgekehrt.
XML-RPC ist für eine Vielzahl an Programmiersprachen verfügbar. Da die CSN-Scripte zum Erstellen der Konfigurationsdateien alle in Perl geschrieben sind, wird auch in dieser Studienarbeit auf die Implementierung in Perl zurückgegriffen.

2.4.2 Vorüberlegungen

Im Moment werden die Konfigurationsdateien über das Perl-Script `userconfig_create` erstellt. Dieses stellt für jeden Dienst, der Konfigurationsdateien benötigt, eine Funktion bereit. Tabelle 2.4.2 listet diese Funktionen kurz auf. Alle Funktionen arbeiten nach einem einheitlichen Schema: Holen der erforderlichen Informationen (meist aus der Datenbank), Aufbau der Konfigurationsdatei in $out und anschließendes Schreiben des Inhalts aus $out in die Datei. Ein anschließender Neustart der Dienste geschieht über das Script `userconfig_execute` und kann ohne größere Anpassungen auf dem Client übernommen werden.

Funktion	**Beschreibung**	**Verzeichnis**
fwconfig();	Firewallregeln	`/var/CSN/firewall/config/*`
dhcpdconfig();	DHCP-Konfiguration	`/etc/dhcpd.conf`
arpconfig();	Router-Konfiguration	`/var/CSN/tftpd/static-arp` `/var/CSN/tftpd/static-arp.diffs`
dsconfig();	Shaper-Konfiguration	`/var/CSN/dynshaper/dynshaper.conf` `/var/CSN/firewall/config/fwmarks`
accountingconfig();	Accounting-Konfiguration	`/var/CSN/accounting/valid_ips`
dnsconfig();	DNS-Konfiguration	`/var/cache/bind/*`

Tabelle 2.4: bereitgestellte Funktionen der Datei userconfig_create

Der Inhalt der Konfigurationsdateien besteht aus mehreren Zeilen American Standard Code for Information Interchange (ASCII) Zeichen. Es kommen also für die Übertragung der Daten nur die beiden XML-RPC-Datenstrukturen `<string>` und `<base64>` in Frage, wobei aufgrund der nicht abschätzbaren Größe der Daten `<string>` gleich wieder verworfen werden kann. Der Pfad und der Dateiname müssen ebenso übertragen werden, hierfür wird die Datenstruktur `<string>` verwendet. Ausserdem ist es bei einigen wenigen Dateien nötig, ihre Zugriffsrechte nach dem Schreiben zu setzen. Hierfür muss der Benutzername (bisher nur 'shaper' für die Shaping-Dateien) mit übertragen werden.
Dieses Triple wird in einem Struct gespeichert, Listing 2.16 zeigt die entsprechende XML-Struktur beispielhaft für die Datei `/var/CSN/dynshaper/dynshaper.conf`. Die Zeilen 2 bis 5 enthalten hier die Pfadangabe samt Dateiname, die Zeilen 6 bis 27 die Konfigurationsdaten und die Zeilen 28 bis 31 sind optional, sie enthalten den Usernamen, falls die Dateirechte abschließend geändert werden müssen.

```
1 <struct>
2   <member>
3     <name>pfad</name>
4     <value><string>/var/CSN/dynshaper/dynshaper.conf</string></
          value>
```

```
5     </member>
6     <member>
7       <name>data</name>
8       <value><base64>
9       #
10      # !!! WARNING !!!
11      # This file is generated automatically by ./
            userconfig_create
12      # Modification is futile!
13      #
14      VERSION="Dynamic␣Traffic␣Shaper␣v0.62"
15      DEVINT="eth0"
16      BWINT="1GBit"
17      DEVEXT="eth1"
18      BWEXT="1GBit"
19      DYNSHAPER="./dynshaper-htb"
20      TC="/var/CSN/bin/tc-helper-script"
21      MODPROBE="/var/CSN/bin/modprobe-helper-script"
22      UGROUPS="1,2,3,4,5,6,7,8,9,10"
23      RATE[1]="8788MBit"
24      IN[1]=""
25      OUT[1]=""
26      </base64></value>
27    </member>
28    <member>
29      <name>chown</name>
30      <value><string>shaper</string></value>
31    </member>
32  </struct>
```

Listing 2.16: Die XML-Struktur für die Konfigurationsdatei ’dynshaper.conf’

Desweiteren werden manchmal mehr als eine Konfigurationsdatei pro Dienst benötigt. Diese müssen aber trotzdem zusammen versendet werden, da wir nur einen Aufruf pro Dienst, und nicht pro Konfigurationsdatei, zur Verfügung haben. Deshalb werden die `<struct>` aus Listing 2.16 zusätzlich in ein `<array>` verpackt, um so mehrere Konfigurationsdateien zu versenden. Listing 2.17 zeigt dies nochmal schematisch.

```
1  <array>
```

```
2     <data>
3       <value>
4         <struct>...Konfig1...</struct>
5       </value>
6       <value>
7         <struct>...Konfig2...</struct>
8       </value>
9     </data>
10  </array>
```

Listing 2.17: Verpacken mehrerer Konfigurationsdateien in einem <array>

2.4.3 Implementierung des Server

Ein XML-RPC-Server ist in Perl mit wenigen Befehlen erstellt. Voraussetzung ist die Installation der `librpc-xml-perl` sowohl auf Client- als auch auf Serverseite. Einen Beispielserver zeigt Listing 2.18. In Zeile 1 werden die benötigten Klassen geladen, Zeile 2 startet einen neuen Server am Port 9000. Mit Hilfe der Methode `add_method` können dem Server Prozeduren hinzugefügt werden, die er den Clients anbietet. Zeile 4 verhindert, dass sich das Script beendet.

```
1     use RPC::XML::Server;
2     $srv = RPC::XML::Server->new(port => 9000);
3     $srv->add_method(...);
4     $srv->server_loop;
```

Listing 2.18: ein einfacher XML-RPC-Server

Das Hinzufügen von Prozeduren (Zeile 3) ist nicht trivial. Es gibt laut Spezifikation [2] drei Möglichkeiten:

- **FILE**
  Die Methode wird aus einer externen Datei geladen, welche in der XPL-Dateistruktur vorliegen muss.
- **HASHREF**
  Angabe einer Hash-Referenz, die die benötigten Informationen über die Methode beinhaltet.
- **LIST**
  Angabe einer Liste mit Informationen über die Methode (name, code, signature, help, version, hidden).

Da die Methoden zum Erstellen der Konfigurationsdateien schon vorliegen, bietet es sich an, die nötigen Informationen als **LIST** zu übergeben. Listing 2.19 zeigt dies anhand der Methode `fwconfig()`. In der Variable $meth_fwconfig wird ein Zeiger auf die Prozedur gespeichert. `name` definiert hier den öffentlichen Namen, den die Clients später rufen. `code` ist eine Referenz auf die eigentliche Perl-Funktion und `signature` stellt eine eindeutige Signatur bereit. Anschließend wird in Zeile 7 die Methode zum Server hinzugefügt.

```
1 my $meth_fwconfig = RPC::XML::Procedure->new({
2         name => 'fwconfig',
3         code => \&fwconfig,
4         signature => [ 'fwconfig' ] }
5 );
6 ...
7 $rpcsrv->add_method($meth_fwconfig);
```

Listing 2.19: Hinzufügen der Methode fwconfig()

Der Server weiß nun, welche Funktionen er bereitzustellen hat, und wie er diese startet. Nun müssen diese Funktionen noch implementiert werden. Die grundlegende Funktionsweise, wie welche Konfigurationsdatei erstellt werden muss, ist schon implementiert und soll hier nicht weiter interessieren. Jede Prozedur ruft mindestens einmal die Funktion `replace_file` auf, welche als Übergabe-Parameter den absoluten Dateinamen und den neuen Inhalt der Konfigurationsdatei erhält. Diese Funktion speichert normalerweise den Inhalt auf die Festplatte, soll nun aber den in Listing 2.16 gezeigten XML-Baum erstellen.
Ein `<struct>` wird mittels `XML::RPC::struct->new()` erstellt, zwischen den Klammern nach `new` können nun einzelne Werte in der Form `'zeigername' => 'Inhalt'` gespeichert werden, wobei Inhalt i.d.R. wieder eine XML::RPC-Datenstuktur ist. Der in Listing 2.20 gezeigte Quelltext erstellt nun unsere `<struct>`-Datenstuktur, die eine Konfigurationsdatei samt Dateipfad und optionalem Usernamen zum Ändern der Rechte enthält. Die Variable $file enthält den Pfad samt Dateiname, $data die Konfigurationsdaten und $changeperm den Usernamen, auf den die Rechte im Anschluss gesetzt werden sollen. Zeile 4 ist dabei optional und wird im Moment nur bei zwei Dateien benötigt (nämlich den beiden Konfigurationsdateien des dynamischen Shapers). Die ganze Struktur wird auf eine Liste (`@filelist`) gepackt, um sie später einfacher zu einem `<array>` zusammenfügen zu können. Listing 2.21 zeigt die dafür geschriebene Funktion `rpc_buildstring`.
Wenn die Prozedur alle Konfigurationsdateien erstellt hat und diese sich auf der `filelist` befinden, wird die Funktion `rpc_returnstring()`(Listing 2.22) aufgerufen, welche die auf der `filelist` befindlichen `<struct>`s in ein `<array>` packt, die `filelist` leert und das `<array>` zum Client schickt.

```
push @filelist, RPC::XML::struct->new(
  'file'  => RPC::XML::string->new($file),
  'data'  => RPC::XML::base64->new($data),
  'chown' => RPC::XML::string->new($change_perm)
);
```

Listing 2.20: Erstellen der struct-Datenstruktur

```
sub rpc_buildstring($$$) {
        my $file = shift;
        my $data = shift;
        my $change_perm = shift;

        if ($change_perm eq 'none') {
          push @filelist, RPC::XML::struct->new(
          'file' => RPC::XML::string->new($file),
          'data' => RPC::XML::base64->new($data)
          );
        } else {
          push @filelist, RPC::XML::struct->new(
          'file' => RPC::XML::string->new($file),
          'data' => RPC::XML::base64->new($data),
          'chown' => RPC::XML::string->new($change_perm)
          );
        }
}
```

Listing 2.21: Die Funktion rpc_buildstring

```
sub rpc_returnstring() {
  my $data = RPC::XML::array->new(@filelist);
  @filelist = ();
  return $data;
}
```

Listing 2.22: Die Funktion rpc_returnstring

Nun ist der Server fertig und betriebsbereit. Der komplette Quelltext ist im Anhang 1 hinterlegt.

2.4.4 Implementierung des Client

Der RPC-Client wird ähnlich wie der Server programmiert. Wieder ist sicherzustellen, dass die Library `librpc-xml-perl` auf dem System installiert ist. Listing 2.23 zeigt einen Beispiel-Client. Zeile 1 erstellt den Client und übergibt ihm die URL sowie den Port zum Server. Anschließend wird ein Request gesendet (Zeile 2), und die auf dem Server zu startende Methode übergeben. Die Antwort wird in $req gespeichert und kann anschließend bearbeitet werden. Hierbei ist sicherzustellen, dass ein möglicher Fehlerfall behandelt wird, um einem inkonsistenten Verhalten vorzubeugen (Zeile 3).

```
1 my $cli = RPC::XML::Client->new('http://localhost:9000/RPCSERV'
    );
2 my $req = $cli->send_request('fwconfig');
3 if ($req->is_fault) {
4   *Fehlerbehandlung*
5 } else {
6   *Programmablauf*
7 }
```

Listing 2.23: ein einfacher XML-RPC-Client

Das Dekodieren der empfangenen Daten geschieht folgendermaßen: die Konfigurationsdateien kommen als `<array>` an und müssen separiert werden. Dies geschieht mit Hilfe der Funktion `handle_data()` (Listing 2.24). Als erstes wird geprüft, ob ein Fehler bei der Verarbeitung aufgetreten ist (Zeile 4) und die Funktion `log_error($fehler)` aufgerufen (Listing 2.25). Ist kein Fehler aufgetreten, wird das Array durchlaufen (Zeile 7) und für jeden Eintrag die Funktion `replace_file($data)` aufgerufen (siehe Listing 2.26). Die Funktion `log_error()` liest die übergebene Fehlernummer samt Fehlermeldung aus (Zeile 4) und gibt sie auf der Konsole aus (Zeile 5). Eine erweiterte Fehlerbehandlung ist hier sicher möglich, aber vorerst nicht von Interesse. Die Funktion `replace_file()` beinhaltet die Routine zum Speichern der Konfigurationsdateien aus der `userconfig_create`. Zu Beginn muss der übergebene `<struct>` aufgespalten werden (Zeile 3 bis 5). In Zeile 7 bis 21 wird die Konfigurationsdatei geschrieben, und Zeile 21 bis 26 setzt, wenn erforderlich, die Rechte der Datei neu.

```
1 sub handle_data($) {
2   my $rpcreq = shift;
3
4   if ($rpcreq->is_fault) {
5     log_error($rpcreq);
6   } else {
7     foreach(@{$rpcreq->value}) {
```

```
8         replace_file($_);
9 }}}
```

Listing 2.24: die Funktion handle_data()

```
1 sub log_error($) {
2   my $fault = shift;
3
4   foreach my $key (keys %{$fault->value}) {
5     print ${$fault->value}{$key}."\n";
6 }}
```

Listing 2.25: die Funktion log_error()

```
1 sub replace_file($) {
2   my $databin = shift;
3   my $data = $databin->{data};
4   my $file = $databin->{file};
5   my $more = $databin->{chown};
6
7   my $tmpl_file = "$file.tmpl";
8   my $out_file  = "$file.new";
9   open OUT, ">$out_file" or die "Cannot␣open␣$out_file␣for␣
        writing:␣$!";
10  if (-f $tmpl_file) {
11    open TMPL, "<$tmpl_file" or die "Cannot␣open␣$tmpl_file␣for
          ␣reading:␣$!";
12    while (<TMPL>) {
13      /^#RULES#/ or print OUT and next;
14      print OUT $data;
15    }
16    close TMPL;
17  } else {
18     print OUT $data;
19  }
20  close OUT;
21  rename "$file.new", "$file" or die "Cannot␣rename␣$file.new␣
        to␣$file:␣$!";
22  if ($more) {
```

```
23      my ($login,$pass,$userid,$gid) = getpwnam($more)
24          or die "User␣'$more'␣not␣in␣passwd␣file";
25      chown $userid, $gid, $file;
26  } }
```

Listing 2.26: die Funktion replace_file()

Der komplette Quelltext des Clients ist im Anhang 2 einzusehen.

2.4.5 Absichern des Systems

Der Server lauscht prinzipiell am definierten Port und stellt jedem Client, der sich in einer für den Server verständlichen Weise ausdrückt, die Konfigurationsdateien zur Verfügung. Das ist natürlich ein ungewollter Zustand, schließlich könnten sich Unbefugte mit Hilfe der Router- oder DHCPD-Konfigurationsdateien ein tieferes Bild der im CSN angeschlossenen Rechner machen.
Der Server muss dahingehend abgesichert werden, dass er nur Verbindungen von (einem) bestimmten Rechner(n) zulässt. Dazu bietet Linux seit Kernel 2.0 eine eigene Firewall, die seit Kernel 2.4 `iptables` heisst. Listing 2.27 zeigt, wie der Serverport mittels `iptables` insofern abgesichert wird, dass er nur Pakete von einer bestimmten IP-Adresse erhält. Dabei steht `-A` für `append` und überprüft Pakete am Ende der Inputlist (`-I INPUT` würde Pakete am Anfang der Inputlist überprüfen). `-p` bezeichnet das Protokoll und `-dport` den Port, an dem der Server lauscht. Die Option `-s` steht für Source und bezeichnet die Quell-IP-Adresse, in unserem Fall also alles, was nicht (! = Negation) 134.109.101.249 (die IP-Adresse des CSN-Servers, auf dem der Client läuft) ist. `-j` ist das sogenannte target und bezeichnet, was mit dem Paket geschehen soll. Hier kann entweder DROP (verwerfen, ohne den Absender zu benachrichtigen) oder REJECT (verwerfen mit Benachrichtigung des Absenders). Um einen potenziellen Angreifer im Unklaren über den Verbleib seiner Pakete zu lassen, empfiehlt sich hier die Option DROP.

```
1 iptables –A INPUT –p tcp ––dport 1663 –s !134.109.101.249 –j
    DROP
```

Listing 2.27: den Server absichern mittels iptables

Das CSN verfügt über Switches der neuesten Generation. Diese sind so konfiguriert, dass jeder MAC-Adresse, die sich ins CSN einklinkt, eine feste IP-Adresse zugeordnet wird. Es ist ferner nicht möglich, sich mit einer anderen als der bei der Anmeldung im CSN zugeordneten IP-Adresse ins Netz zu verbinden, da der Switch die Kombination überprüft und bei Fehlkonfigurationen am Nutzerrechner, seien sie beabsichtigt oder nicht, sofort den Port sperrt. Insofern ist im CSN das so genannte IP-Spoofing, also das Versenden von IP-Paketen mit gefälschter Quell-Adresse, nicht möglich. Es ist weiterhin auch nicht möglich, einen Man-in-the-Middle-Angriff, bei dem der Angreifer zwischen Client und Server

steht, und den gesamten Datenverkehr mitlauscht, durchzuführen. Somit ist die `Iptables`-Regel als sicher einzustufen.
Ein externer Zugriff auf den Server ist auch nicht möglich, da die Firewall nur Verbindungen von extern nach intern zulässt, wenn der Verbindungsaufbau vom internen Rechner gestartet wurde.
Eine weitere Möglichkeit, den Datenverkehr zwischen Server und Client abzusichern, besteht in der Verwendung der im CSN vorhandenen VLANS. So existiert im gesamten CSN ein Management-VLAN, über das u.a. die Switches konfiguriert werden. Es ist also anzuraten, die Switches so zu konfigurieren, dass sie den Datenverkehr zwischen XML-RPC-Server und -Client über dieses Management-VLAN versenden.

2.4.6 Vor- und Nachteile

Der enorme Vorteil ist, dass keine zusätzlichen Tools verwendet und vor allem konfiguriert werden müssen, als sowieso schon verwendet werden. Man kann die XML-RPC-Variante auch ohne weiteres erweitern, indem man einfach ein kleines Perl-Script schreibt, in dem man einen weiteren Client erstellt, der auf einem dritten (oder vierten) Rechner läuft. Einer weiteren Verteilung der Dienste steht hiermit also nichts im Weg. Aufgepasst werden muss bei der Sicherheit. In einem Testbetrieb sollte die Sicherheit der iptables-Regeln nochmals genau untersucht werden.

Vorteile	**Nachteile**
ausschließliche Nutzung vorhandener Ressourcen	Sicherheit genau überprüfen
kein zusätzlicher Konfigurationsaufwand	
beliebig, einfach und schnell erweiterbar	

Tabelle 2.5: Vor- und Nachteile Variante XML-RPC

3 Migrationsstrategie

3.1 Zielsetzung

Durch eine geeignete Migrationsstrategie können die Ausfallzeiten während und nach dem Umbau minimiert werden. Ziel sollte es sein, die Variante zu finden, bei der die externe Anbindung so kurz wie möglich unerreichbar ist. Ausserdem sollten unvorhersehbare Ereignisse während der Arbeit die Ausfallzeit nicht unnötig verlängern. Dies könnte z.B. dadurch bewerkstelligt werden, dass eine vorher installierte Testimplementation kurfristig als Backuplösung eingesetzt wird. Doch Näheres dazu im Folgenden.

3.2 Step-by-Step: Umsetzung der XML-RPC-Variante

Nach Rücksprache mit einigen CSN-Mitgliedern haben wir uns darauf geeinigt, dass die XML-RPC-Variante in Zukunft im CSN eingesetzt werden soll. Sie besticht vor allem durch Ihre Flexibilität und die Tatsache, dass sie direkt in die schon vorhandenen Perl-Scripte implementiert wurde.

3.2.1 Installation der Firewall

Zu Beginn sollte der neue CSN-Server gekauft werden. Ein Test der Funktionstüchtigkeit der Hardwarekomponenten ist unumgänglich (RAM testen, Festplatte auf fehlerhafte Sektoren überprüfen, . . .), um Überraschungen während oder nach der Installation vorzubeugen. Anschließend kann mit der Installation des Betriebssystems (Debian 3.1) begonnen werden. Danach nicht benötigte Dienste deaktivieren und die aktuellen Konfigurationsdateien installieren. Nun müssen noch alle Scripte einschließlich des RPC-Client gespeichert werden. Sind alle Dienste gestartet und augenscheinlich richtig konfiguriert, kann im CSN-Labor ein Testlauf gestartet werden.
Alle bisherigen Aktionen hatten keinen Ausfall der externen Anbindung im CSN zur Folge.

3.2.2 Änderungen an der Datenbank

Um einen längen Ausfall und eine im Fall der Fälle möglicherweise lange Fehlersuche (die berühmte ’Nadel im Heuhaufen’) zu unterbinden, schlage ich vor, den alten CSN-Server nicht neu aufzusetzen, sondern die darauf laufende Datenbank für den ersten Betrieb zu nutzen. Es sind nur ein paar Änderungen zu machen, die im Folgenden erläutert werden. Da die Datenbank nun auch von extern erreichbar

sein muss, ist die Konfiguration entsprechend anzupassen. PostgreSQL lauscht standardmäßig am Port 5432, die Firewall ist so zu konfigurieren, dass sie an diesem Port nur Verbindungen von internen CSN-Rechnern zulässt. Desweiteren benötigt der alte CSN-Server eine neue IP-Adresse und seinen neuen Namen `db1.csn.tu-chemnitz.de`. Zum Schluß werden die weiteren Dienste deaktiviert, da sie ja nun auf dem neuen CSN-Server laufen. Sie können im Fehlerfall als zusätzliche Informationsquelle genutzt werden, um Fehler beim Übertragen der Konfiguration auszuschließen. Nun ist der neue DB-Server in die Vetterstrasse zu tragen und dort einzubauen. Vorher kann der neue CSN-Server schon in Betrieb genommen werden, denn die meisten Dienste sollten auch ohne aktiven DB-Server laufen, da die initialen Konfigurationsdateien ja schon manuell installiert wurden. Somit kann die Zeit des Netzausfalls minimiert werden.
Nach Einbau des DB-Servers in der Vetterstrasse kann der RPC-Server installiert und die Funktionen des DB-Servers vom CSN-Labor aus geprüft werden. Erst wenn sicher ist, dass der RPC-Server die richtigen Konfigurationsdateien ausliefert, kann der RPC-Client auf dem CSN-Server gestartet und im Erfolgsfall in die crontab geschrieben werden.
Nun sind noch die anderen CSN-Scripte auf dem CSN-Server so anzupassen, dass sie zukünftig nicht mehr lokal sondern über das Netzwerk auf den neuen Datenbank-Server zugreifen.

4 Ausblick: Hochverfügbarkeit

Diese Studienarbeit dient dazu, dem CSN Vorschläge zu machen, wie die Situation um den CSN-Server als die 'eierlegende Wollmilchsau' gelockert werden kann. Es wurden verschiedene Möglichkeiten aufgezeigt, wie eine Trennung von Diensten und Datenbasis aussehen kann. Doch auch ein Auslagern der Datenbank auf einen neuen Server verhindert nicht, dass der CSN-Server und die Datenbank Single Point of Failure (SPoF) Komponenten im StudentenNetz sind.
Um auch in Zukunft die Ausfallwahrscheinlichkeit und -dauer des Netzes so gering wie möglich zu halten, müssen weiterführende Ideen ins CSN integriert werden. Zum Beispiel: Ein zweiter Dienste- und Datenbank-Rechner, die als Slave arbeiten und einspringen, sobald einer der Masterrechner ausfällt, würde einen Komplettausfall des Netzes sehr unwahrscheinlich machen und damit die Hochverfügbarkeit der Dienste im CSN steigern.
Für ProstgreSQL gibt es schon fertige Implementierungen, allen voran Slony [3]. Die Dienste-Rechner müssten sich nun untereinander die Konfigurationsdateien auf dem aktuellsten Stand halten, dies sollte mit Hilfe der in dieser Studienarbeit vorgeschlagenen XML-RPC-Variante keine Hürde darstellen.
Ein CSN, das nicht mehr nur über einen Server (aktueller Stand), sondern einen replizierten Dienste- und einen replizierten Datenbankerver verfügt, ist in Punkto Hochverfügbarkeit und Lastverteilung auf Jahre hinaus gesichert.

Literaturverzeichnis

[1] *Extensible Markup Language - Wikipedia*.
URL `http://de.wikipedia.org/wiki/XML`

[2] *Implementation of XML-RPC in Perl*.
URL `http://www.blackperl.com/RPC::XML/`

[3] *Slony-I — A replication system for PostgreSQL*.
URL `http://gborg.postgresql.org/project/slony1/projdisplay.php`

[4] Intel Corporation: *Communication Streaming Architecture - Reducing the PCI Network Bottlenet*.
URL `http://www.intel.com/design/network/papers/25245102.pdf`

[5] PostgreSQL FAQ: *What computer hardware should I use?*.
URL `http://www.postgresql.org/docs/faqs.FAQ.html#item3.7`

[6] UserLand Software: *XML-RPC Spezifikation*.
URL `http://xmlrpc.com/spec`

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 31. Juli 2006

_____

Sebastian Wehrmann

Anhang

```
1  #!/usr/bin/perl -w -T
2  ##############################################################################
3  #
4  # userconfig_create (C) 2001-2005 by
5  #                       Tilo Buschmann,
6  #                       Heiko Jehmlich,
7  #                       Christian Krause,
8  #                       Mirko Parthey,
9  #                       Markus Schade,
10 #                       Sebastian Wehrmann
11 #
12 # description:  erzeugt nutzerbasierte Configfiles:
13 #               - firewall
14 #               - dynshaper
15 #               - static arp
16 #               - dhcpd
17 #               - tcp-resetter
18 #               - dns
19 #
20 # version:      $Id: userconfig_create 6600 2006-07-12 12:46:40Z sweh $
21 #
22 # license:      General Public License version 2
23 #
24 ##############################################################################
25
26 use strict;
27 use DBI;
28 use POSIX qw(strftime);
29 use CSN;
30 use RPC::XML::Server;
31
32 sub fwconfig();
33 sub dhcpdconfig();
34 sub arpconfig();
35 sub dsconfig();
36 sub macconfig();
37 sub accountingconfig();
38 sub dnsconfig();
39
40 sub get_hostpart($);
41 sub get_subnet($);
42 sub get_c_subnet($);
43 sub replace_file($$);
44 sub rpc_buildstring($$$);
45 sub rpc_returnstring();
46 sub ip_pack($);
47 sub ip_unpack($);
48 sub subnet_enum_hosts($$);
49 sub reformat_mac($);
50
51 # Globale Defaults
52 ##############################################################################
```

```
53
54 my $RPCHOST = "localhost";
55 my $RPCPORT = "1663";
56
57 my $INVALID_MAC = "14:d5:d9:bc:ae:57";
58 my $MASTER = 'csn-server.csn.tu-chemnitz.de.';
59 my $MASTERIP = "134.109.101.249";
60
61 my $FWBASE = "/var/CSN/firewall";
62 my $ZONEBASE ="/var/cache/bind";
63 my $DHCPDCONF = "/etc/dhcpd.conf";
64 my $ARPCONF = "/var/CSN/tftpd/static-arp";
65 my $ARP_DIFFS = "/var/CSN/tftpd/static-arp.diffs";
66 my $DSCONF="/var/CSN/dynshaper/dynshaper.conf";
67 my $MACCONF = "/var/CSN/tftpd/macs";
68 my $ACCOUNTINGCONF = "/var/CSN/accounting/valid_ips";
69 my $FORWARDCONF = "db.csn.tu-chemnitz.de";
70 my %REVERSECONF;
71 my $DNSCHANGE = "/var/run/bind/changed";
72
73 my @MX = ('john.hrz.tu-chemnitz.de.','lana.hrz.tu-chemnitz.de.');
74 my @DNS = ('obelix.hrz.tu-chemnitz.de.', 'saturn.hrz.tu-chemnitz.de.');
75 my $NNTPHOST = "anderson.hrz.tu-chemnitz.de.";
76 my @NTPHOSTS = ('zuse.hrz.tu-chemnitz.de.', 'phoenix.hrz.tu-chemnitz.de.');
77
78 my %ADMIN_HOST = (
79         zensiert => {
80                 ip      => "xxx.xxx.xxx.xxx",
81                 mac     => "xx:xx:xx:xx:xx:xx"
82         }
83 );
84
85 # begin
86 ##########################################################################
87
88 my $order = {};
89 my (%oper, %hosts);
90 my @team_ips;
91 my @filelist;   # äenthlt die Konfigfiles üfr den RPC-Server
92
93 @team_ips = get_Aufgabengebiet_IPs('core-team');
94 my $dbh = db_connect("firewall_conf");
95
96 # View ohne Trafficwerte
97 # (weil "-j MARK" sowieso nicht das Ende der Bearbeitung ist)
98 #
99 my $sth = db_call($dbh, "SELECT * FROM firewall_config");
100 while (my $r = $sth->fetchrow_hashref()) {
101         my $ip = $r->{ip_adr};
102         $hosts{$ip}{mac} = $r->{ether_adr};
103         $hosts{$ip}{hostname} = $r->{hostname};
104         my $sn = get_subnet($ip);
```

```
105             push @{$order->{$sn}}, $ip; # hirnrissig -> $hosts
106         }
107         $sth->finish();
108         $dbh->disconnect();
109
110 # overwrite admin hosts
111 foreach my $host (keys %ADMIN_HOST) {
112   my $ip = $ADMIN_HOST{$host}{ip};
113   $hosts{$ip}{mac} = $ADMIN_HOST{$host}{mac};
114   $hosts{$ip}{hostname} = $host;
115 }
116
117 # Initialisierung des RPC-Servers
118 # 07/2006 by sweh
119 #
120
121 # Methoden, auf die der Server reagistert
122 my $meth_fwconfig = RPC::XML::Procedure->new({
123         name => 'fwconfig',
124         code => \&fwconfig,
125         signature => [ 'fwconfig' ] }
126 );
127 my $meth_dhcpdconfig = RPC::XML::Procedure->new({
128         name => 'dhcpdconfig',
129         code => \&dhcpdconfig,
130         signature => [ 'dhcpdconfig' ] }
131 );
132 my $meth_arpconfig = RPC::XML::Procedure->new({
133         name => 'arpconfig',
134         code => \&arpconfig,
135         signature => [ 'arpconfig' ] }
136 );
137 my $meth_dsconfig = RPC::XML::Procedure->new({
138         name => 'dsconfig',
139         code => \&dsconfig,
140         signature => [ 'dsconfig' ] }
141 );
142 my $meth_macconfig = RPC::XML::Procedure->new({
143         name => 'macconfig',
144         code => \&macconfig,
145         signature => [ 'macconfig' ] }
146 );
147 my $meth_accountingconfig = RPC::XML::Procedure->new({
148         name => 'accountingconfig',
149         code => \&accountingconfig,
150         signature => [ 'accountingconfig' ] }
151 );
152 my $meth_dnsconfig = RPC::XML::Procedure->new({
153         name => 'dnsconfig',
154         code => \&dnsconfig,
155         signature => [ 'dnsconfig' ] }
156 );
```

```
157
158 # einen Server starten
159 my $rpcsrv = RPC::XML::Server->new(host => $RPCHOST, port => $RPCPORT);
160
161 # Methoden hinzufuegen
162 $rpcsrv->add_method($meth_fwconfig);
163 $rpcsrv->add_method($meth_dhcpdconfig);
164 $rpcsrv->add_method($meth_arpconfig);
165 $rpcsrv->add_method($meth_dsconfig);
166 $rpcsrv->add_method($meth_macconfig);
167 $rpcsrv->add_method($meth_accountingconfig);
168 $rpcsrv->add_method($meth_dnsconfig);
169
170 # erforderlich, damit der Server nie return
171 $rpcsrv->server_loop;
172
173 # end
174 ##########################################################################
175 # begin subroutines
176
177 #
178 # Alle gueltigen IP Adressen fuer Gesamttraffic (router-Traffic)
179 # 05/2004 by hje
180 #
181 sub accountingconfig() {
182         my $out =       "#\n# !!! WARNING !!!\n" .
183                 "# This file is generated automatically by $0\n" .
184                 "# Modification is futile!\n#\n";
185
186         foreach my $sn (80, 84, 88, 92, 96, 104, 108, 112, 114, 116) {
187                 foreach my $ip (sort sort_by_ip @{$order->{$sn}}) {
188                         $out .= "$ip\n";
189                 }
190         }
191         rpc_buildstring($ACCOUNTINGCONF, $out, 'none');
192         return rpc_returnstring();
193 }
194
195 #
196 # MAC-Adressen fuer den TCP-Resetter in den BNC Wohnheimen
197 # 10/2003 by hje
198 #
199 sub macconfig() {
200         foreach my $sn (104, 116) {
201                 my $out = "";
202                 foreach my $ip (sort sort_by_ip @{$order->{$sn}}) {
203                         $out .= "$ip $hosts{$ip}{mac}\n";
204                 }
205                 rpc_buildstring("$MACCONF.$sn", $out, 'none');
206         }
207         return rpc_returnstring();
208 }
```

```
209
210 #
211 # Firewall config erzeugen
212 # 09/2003 by hje
213 #
214 sub fwconfig() {
215         my @SUBNETS = ( 80, 84, 88, 92, 96, 102, 103, 104, 108, 112, 114, 116 );
216
217         mkdir "$FWBASE/config" if not -d "$FWBASE/config";
218
219         my $out =       "#\n# !!! WARNING !!!\n" .
220                 "# This file is generated automatically by $0\n" .
221                 "# Modification is futile!\n#\n";
222
223         foreach my $ip (@team_ips) {
224                 $out .= "$ip\n";
225         }
226
227         $out .= "134.109.110.0/23\n"; # Lab hat denselben Zugriff wie Team
228         rpc_buildstring("$FWBASE/config/hosts.core_team", $out, 'none');
229
230         $out =  "#\n# !!! WARNING !!!\n" .
231                 "# This file is generated automatically by $0\n" .
232                 "# Modification is futile!\n#\n";
233
234         foreach my $host (keys %ADMIN_HOST) {
235                 $out .= $ADMIN_HOST{$host}{ip} . "\n";
236         }
237         rpc_buildstring("$FWBASE/config/hosts.admin", $out, 'none');
238
239         foreach my $sn (@SUBNETS) {
240                 $out =  "#\n# !!! WARNING !!!\n" .
241                         "# This file is generated automatically by $0\n" .
242                         "# Modification is futile!\n#\n";
243
244                 foreach my $ip (@{$order->{$sn}}) {
245                         $out .= "$ip\n";
246                 }
247                 rpc_buildstring("$FWBASE/config/order.$sn", $out, 'none');
248         }
249         return rpc_returnstring();
250 }
251
252 #
253 # DHCPD config file erzeugen
254 # 06/2003 by tibu
255 #
256 sub dhcpdconfig() {
257         # host pinetree {
258         #         hardware ethernet 00:08:c7:aa:e5:75;
259         #         fixed-address 134.109.80.5;
260         #         option host-name "pinetree";
```

```
261         # }
262
263         my $out =   "#\n#␣!!!␣WARNING␣!!!\n" .
264                     "#␣This␣file␣is␣generated␣automatically␣by␣$0\n" .
265                     "#␣Modification␣is␣futile!\n#\n";
266
267         foreach my $ip (sort sort_by_ip keys %hosts) {
268                 if (
269                         (defined $hosts{$ip}) and
270                         (defined $hosts{$ip}{'hostname'}) and
271                         ($hosts{$ip}{'hostname'} ne '')
272                    ) {
273                         my $mac = $hosts{$ip}{mac};
274                         $out .= "host␣$hosts{$ip}{'hostname'}␣{\n";
275                         $out .= "\thardware␣ethernet␣$mac;\n";
276                         $out .= "\tfixed-address␣$ip;\n";
277                         $out .= "\toption␣host-name␣\"$hosts{$ip}{'hostname'}\";\n";
278                         $out .= "}\n";
279                 }
280         }
281         rpc_buildstring($DHCPDCONF, $out, 'none');
282         return rpc_returnstring();
283 }
284
285 #
286 # Router config file erzeugen
287 # 04/2003 by chkr, mpa
288 # rewritten 09/2003 by hje
289 #
290 sub arpconfig() {
291         my $new_mac;
292         my $out = "";
293         my $diffs = "";
294         my %old_mac;
295         my @SUBNETS = ( 80, 84, 88, 92, 96, 104, 108, 112, 114, 116 );
296
297         # alte config einlesen
298         open OLD, "<$ARPCONF" or die "Cannot␣open␣$ARPCONF␣for␣reading:␣$!";
299         while (<OLD>) {
300                 chomp;
301                 next if /^end/;
302                 my @parts = split(/ /, $_);
303                 $old_mac{$parts[1]} = $parts[2];
304         }
305         close OLD;
306
307         foreach my $sn (@SUBNETS) {
308                 # generate all hosts
309                 my @hosts = subnet_enum_hosts("134.109.$sn.0", "255.255.254.0");
310
311                 my $gw = sprintf("134.109.%d.254", $sn+1);
312                 foreach my $ip (@hosts) {
```

```
313                         next if $ip eq $gw;
314                         if (defined $hosts{$ip}{mac}) {
315                                 $new_mac = $hosts{$ip}{mac};
316                         } else {
317                                 $new_mac = $INVALID_MAC;
318                         }
319                         $out .= "arp␣$ip␣" . reformat_mac($new_mac) . "␣ARPA\n";
320                         $diffs .= "arp␣$ip␣" . reformat_mac($new_mac) . "␣ARPA\n"
321                             if $old_mac{$ip} ne reformat_mac($new_mac);
322                 }
323         }
324         $out .= "end\n";
325         $diffs .= "end\n" if $diffs ne "";
326
327         unlink $ARP_DIFFS if -f $ARP_DIFFS;
328         rpc_buildstring($ARP_DIFFS, $diffs, 'none') if $diffs ne "";
329         rpc_buildstring($ARPCONF, $out, 'none');
330         replace_file($ARPCONF, $out);  #weil lokal öbentigt
331         return rpc_returnstring();
332 }
333
334 #
335 # Dynshaper config file und fwmarks fuer firewall erzeugen
336 # 08/2003 by marks
337 #
338 sub dsconfig() {
339         my $dbh = db_connect("traffic_shaper");
340         my (%fwconf, %dsallg, %group_info, %excepts);
341         my ($sql, $sth, $out);
342
343         # Allgemeine Dynshaper Parameter holen
344         $sql = qq{SELECT * from ds_allgemein_v2};
345         $sth = db_call($dbh,$sql);
346         while (my $r = $sth->fetchrow_hashref()) {
347                 $dsallg{$r->{parameter}} = $r->{wert};
348         }
349         $sth->finish;
350
351         # Gruppenparameter holen
352         $sql = qq{SELECT * from ds_gruppen_v2};
353         $sth = db_call($dbh,$sql);
354         while (my $r = $sth->fetchrow_hashref()) {
355                 $group_info{$r->{gruppe}}{$r->{parameter}} = $r->{wert};
356         }
357         $sth->finish;
358
359         $sql = qq{SELECT * from ds_ausnahmen_v2};
360         $sth = db_call($dbh,$sql);
361         while (my $r = $sth->fetchrow_hashref()) {
362                 $excepts{$r->{ausnahme}}{$r->{parameter}} = $r->{wert};
363         }
364         $sth->finish;
```

```
365
366         # IPs einer zu einer person_id holen (auch ü fr die fw)
367         $sql = qq{SELECT * FROM firewall_config
368                     WHERE person_id IS NOT NULL
369                     AND manglemask IS NOT NULL};
370
371         $sth = db_call($dbh,$sql);
372         while (my $r = $sth->fetchrow_hashref()) {
373                 # fwmark aus person_id und Netzklassenbitmaske berechnen
374                 my $mark = int($r->{person_id}) | int($r->{manglemask});
375                 # shaping gruppe der person_id
376                 $fwconf{$r->{person_id}}{gruppe} = $r->{gruppe};
377                 # fwmark per IP
378                 $fwconf{$r->{person_id}}{$r->{ip_adr}} = $mark;
379                 # fwmarks einer person_id
380                 push(@{$fwconf{$r->{person_id}}{mark}},$mark);
381                 # IP Adressen einer person_id
382                 push(@{$fwconf{$r->{person_id}}{ip}},$r->{ip_adr});
383         };
384         $sth->finish();
385         $dbh->disconnect();
386
387
388         ##################### Shaper Config File generieren ################
389         $out =  "#\n# !!! WARNING !!!\n" .
390                 "# This file is generated automatically by $0\n" .
391                 "# Modification is futile!\n#\n";
392
393         $out .= "VERSION=\"Dynamic Traffic Shaper v0.62\"\n";
394         $out .= "DEVINT=\"" . $dsallg{conf_devint} . "\"\n";
395         $out .= "BWINT=\"" . format_bits($dsallg{conf_bwint}) . "\"\n";
396
397         $out .= "DEVEXT=\"" . $dsallg{conf_devext} . "\"\n";
398         $out .= "BWEXT=\"" . format_bits($dsallg{conf_bwext}) . "\"\n";
399
400
401         $out .= "DYNSHAPER=\"" . $dsallg{conf_dspath} . "\"\n";
402         $out .= "TC=\"" . $dsallg{conf_tcpath} . "\"\n";
403         $out .= "MODPROBE=\"" . $dsallg{conf_mppath} . "\"\n";
404
405         #Gruppenliste
406         $out .= "UGROUPS=\"";
407         foreach my $group (sort {$a <=> $b} keys %group_info)
408         {
409                 $out .= "$group,"
410         }
411         chop $out; $out .= "\"\n";
412
413
414         foreach my $group (sort {$a <=> $b} keys %group_info)
415         {
```

```
416             my $rate = $group_info{$group}{conf_rate} * $group_info{$group}{conf_factor
                    };
417
418             # BB jeder Gruppe
419             $out .= "RATE[$group]=\"" . format_bits_fine($rate) ."\"\n";
420
421             # FIXME: Minimum BB der beiden Interfaces zum Vergleich nutzen
422             # Perl hat keinen Min/Max Operator
423
424             # To shape or not to shape (incoming)
425             if ($rate > $dsallg{conf_noshape} || $rate > $dsallg{conf_bwint} || !
                    $group_info{$group}{conf_in})
426             {
427                     $out .= "IN[$group]=\"\"\n";
428             } else {
429                     $out .= "IN[$group]=\"on\"\n";
430             }
431
432             # to shape or not to shape (outgoing)
433             if ($rate > $dsallg{conf_noshape} || $rate > $dsallg{conf_bwext} || !
                    $group_info{$group}{conf_out})
434             {
435                 $out .= "OUT[$group]=\"\"\n";
436             } else {
437                 $out .= "OUT[$group]=\"on\"\n";
438             }
439
440             # Pro Gruppe die FW-Marks schreiben
441             $out .= "MARKS[$group]=\"";
442             foreach my $uid (keys %fwconf) {
443                     if ($fwconf{$uid}{gruppe} == $group) {
444                             $out .= "$uid:";
445                             $out .= join (",", @{$fwconf{$uid}{mark}});
446                             $out .= ";";
447                     }
448             }
449             chop $out; $out .= "\"\n";
450         }
451
452         # und zum schluss die Ausnahmen
453         $out .= "EXCEPTS=\"";
454         foreach my $case (sort {$a <=> $b} keys %excepts) {
455                 $out .= "$case␣"
456         }
457         chop $out; $out .= "\"\n";
458
459         foreach my $case (sort {$a <=> $b} keys %excepts) {
460                 $out .= "ERATE[$case]=\"" . format_bits($excepts{$case}{conf_rate}) ."\"\n"
                        ;
461                 $out .= "EPRIO[$case]=\"" . $excepts{$case}{conf_prio} ."\"\n";
462
463                 # Mehrere Matches stehen in der DB mit , getrennt
```

```
464                 # Der tc filter will aber ; haben
465                 $excepts{$case}{conf_in} =~ s/,/;/g;
466                 $out .= "EIN[$case]=\"" . $excepts{$case}{conf_in} ."\"\n";
467
468                 # Mehrere Matches stehen in der DB mit , getrennt
469                 # Der tc filter will aber ; haben
470                 $excepts{$case}{conf_out} =~ s/,/;/g;
471                 $out .= "EOUT[$case]=\"" . $excepts{$case}{conf_out} ."\"\n";
472                 $out .= "EBOUND[$case]=\"" . $excepts{$case}{conf_bound} ."\"\n";
473
474         }
475
476         rpc_buildstring($DSCONF, $out, 'shaper');
477         # chown shaper.root $DSCONF
478         #my ($login,$pass,$userid,$gid) = getpwnam('shaper')
479         #    or die "User 'shaper' not in passwd file";
480         #chown $userid, $gid, $DSCONF;
481
482
483         ##################### fwmarks Config fuer Firewall generieren ################
484         $out =  "#\n#␣!!!␣WARNING␣!!!\n" .
485                 "#␣This␣file␣is␣generated␣automatically␣by␣$0\n" .
486                 "#␣Modification␣is␣futile!\n#\n";
487         foreach my $uid (sort keys %fwconf) {
488                 foreach my $ip (@{$fwconf{$uid}{ip}}) {
489                         $out .= "$ip:$fwconf{$uid}{$ip}\n";
490                 }
491         }
492         rpc_buildstring("$FWBASE/config/fwmarks", $out, 'shaper');
493         #chown $userid, $gid, "$FWBASE/config/fwmarks";
494         return rpc_returnstring();
495 }
496
497 #
498 # Zonefile Generierung fuer den Bind
499 # 2005 by marks
500 #
501 sub dnsconfig () {
502         my $dbh = db_connect('dns');
503         my ($sql, $sth);
504
505         my $soa ='';
506         my $serial = strftime "%Y%m%d", localtime;
507         my $revision;
508         my $forward = '';
509
510         my %hosts;
511         my %ips;
512
513         $sql = qq{SELECT count(*) as count
514                   FROM rechner
515                   WHERE ether_adr IS NOT NULL
```

```
516                 AND ip_adr IS NOT NULL
517                 AND last_change
518                 BETWEEN ('now'::abstime - '1 hour'::interval)
519                 AND 'now'::abstime};
520
521         $sth = db_call($dbh,$sql);
522
523         my $row = $sth->fetchrow_hashref();
524         $sth->finish();
525
526         unlink $DNSCHANGE;
527
528         if ($row->{"count"} != 0) {
529                 rpc_buildstring($DNSCHANGE, $serial, 'none');
530         } else {
531                 $dbh->disconnect();
532                 return;
533         }
534
535         # Vergebene IPs aus DB holen
536         $sql = qq{SELECT hostname, ip_adr
537                 FROM rechner
538                 WHERE ip_adr IS NOT NULL
539                 ORDER BY hostname};
540
541         $sth = db_call($dbh,$sql);
542
543         while (my $r = $sth->fetchrow_hashref()) {
544                 $hosts{$r->{hostname}} = $r->{ip_adr};
545                 $ips{$r->{ip_adr}} = $r->{hostname};
546         }
547         $sth->finish();
548
549         # Letzte Serial aus DB auslesen
550         $sql = qq{SELECT max(revision) from dnsupdate where datum='$serial'};
551         $sth = db_call($dbh,$sql);
552         $revision = $sth->fetchrow_array();
553
554         if (defined $revision) {
555                 $revision++;
556         } else {
557                 $revision = 0;
558         }
559
560         if ($revision < 10) {
561             $serial .= 0;
562         }
563         $serial .= $revision;
564
565         $soa .="\t\t\t\t$serial\t; serial\n";
566         $soa .="\t\t\t\t10800\t\t; refresh (3 hours)\n";
567         $soa .="\t\t\t\t3600\t\t; retry (1 hour)\n";
```

```
568         $soa .="\t\t\t\t3600000\t\t; expire (5 weeks 6 days 16 hours)\n";
569         $soa .="\t\t\t\t3600\t\t; minimum (1 hour)\n";
570         $soa .="\t\t\t\t)\n";
571 
572         foreach my $dnssrv (@DNS) {
573             $soa .="\t\t\tNS\t$dnssrv\n";
574         }
575 
576         #
577         # Nun beginnen wir das FORWARD Zonefile zu erzeugen
578         #
579 
580         $forward = "\$TTL 3600\n";
581         $forward .= "csn.tu-chemnitz.de.\tIN SOA  $MASTER ";
582         $forward .="hostmaster.csn.tu-chemnitz.de. (\n";
583         $forward .= $soa;
584 
585         $forward .="\t\t\tA\t$MASTERIP\n";
586 
587         foreach my $mail (@MX) {
588             $forward .= "\t\t\tMX\t1 $mail\n";
589         }
590 
591 
592         #
593         # Jetzt üfgen wir die A Records ein
594         #
595         $forward .="; Beginn CSN Hosts\n";
596         $forward .="\$TTL 86000\n";
597 
598         # Damit auch localhost.csn auf localhost zeigt
599         $forward .= "localhost\t\tA\t127.0.0.1\n";
600 
601         foreach my $hostname (sort {$a cmp $b} keys %hosts) {
602             # Was tut man nicht alles fuer ein schoenes Zonefile
603             if (length($hostname) >= 16) {
604                 $forward .= "$hostname\tA\t$hosts{$hostname}\n";
605             } elsif (length($hostname) >= 8) {
606                 $forward .= "$hostname\t\tA\t$hosts{$hostname}\n";
607             } else {
608                 $forward .= "$hostname\t\t\tA\t$hosts{$hostname}\n";
609             }
610         }
611 
612         #
613         # Anschliessend kommen die CNAMES
614         #
615 
616         $forward .= "; Beginn CNAMES\n";
617         $forward .= "ftp\t\t\tCNAME\tftpl.tu-chemnitz.de.\n";
618         $forward .= "loghost\t\t\tCNAME\tlocalhost\n";
619         $forward .= "mailhost\t\tCNAME\tmailhost.tu-chemnitz.de.\n";
```

```
620         $forward .= "nntphost\t\tCNAME\t$NNTPHOST\n";
621
622         # TIMESERVER
623         my $ntpnr = 1;
624         foreach my $ntphost (@NTPHOSTS) {
625                 $forward .= "ntphost$ntpnr\t\tCNAME\t$ntphost\n";
626                 $ntpnr++;
627         }
628
629
630         $sql = qq{SELECT alias, hostname
631                   FROM cnames c
632                   INNER JOIN rechner r ON c.host_id=r.host_id};
633
634         $sth = db_call($dbh,$sql);
635
636         while (my $r=$sth->fetchrow_hashref()) {
637                 if (length($r->{alias}) >= 16) {
638                         $forward .= "$r->{alias}\tCNAME\t$r->{hostname}\n";
639                 } elsif (length($r->{alias}) >= 8) {
640                         $forward .= "$r->{alias}\t\tCNAME\t$r->{hostname}\n";
641                 } else {
642                         $forward .= "$r->{alias}\t\t\tCNAME\t$r->{hostname}\n";
643                 }
644         }
645
646         $sth->finish();
647
648         #
649         # äVollstndiges Forward File austauschen
650         #
651         rpc_buildstring("$ZONEBASE/$FORWARDCONF", $forward, 'none');
652
653         #
654         # Jetzt Reverse Zones erzeugen
655         #
656         my %snips;
657
658         foreach my $ip (sort sort_by_ip keys %ips) {
659             my $subnet = get_c_subnet($ip);
660             $snips{$subnet}{$ip} = $ips{$ip} . ".csn.tu-chemnitz.de.";
661         }
662
663         foreach my $sn (80 .. 99, 101 .. 119) {
664                 $REVERSECONF{$sn} .= "\$ORIGIN .\n";
665                 $REVERSECONF{$sn} .= "\$TTL 86000\n";
666                 $REVERSECONF{$sn} .= "$sn.109.134.in-addr.arpa\tIN SOA";
667                 $REVERSECONF{$sn} .= "  $MASTER ";
668                 $REVERSECONF{$sn} .= "hostmaster.csn.tu-chemnitz.de. (\n";
669                 $REVERSECONF{$sn} .= $soa;
670                 $REVERSECONF{$sn} .= "; Begin Reverse Zone\n";
671                 $REVERSECONF{$sn} .= "\$ORIGIN $sn.109.134.in-addr.arpa.\n";
```

```
672                 # DNS Name des Subnetzes
673                 $sql = qq(SELECT dnsname from subnetz WHERE netz_adr >> '134.109.$sn.0');
674                 $sth = db_call($dbh,$sql);
675                 my $sname = $sth->fetchrow_array();
676                 $sth->finish();
677                 $sname .= '.csn.tu-chemnitz.de.' if defined $sname;
678                 $REVERSECONF{$sn} .= "0\t\t\tPTR\t$sname\n" if defined $sname;
679                 # Jetzt die IPs hinzufuegen
680                 foreach my $ip (sort sort_by_ip keys %{$snips{$sn}}) {
681                     $REVERSECONF{$sn} .= get_hostpart($ip);
682                     $REVERSECONF{$sn} .= "\t\t\tPTR\t$snips{$sn}{$ip}\n";
683                 }
684                 if ($sn == 101) {
685                     $REVERSECONF{$sn} .= "252\t\t\tPTR\tc6-39-027-1.hrz.tu-chemnitz.de.\n";
686                     $REVERSECONF{$sn} .= "253\t\t\tPTR\tc6-39-027-2.hrz.tu-chemnitz.de.\n";
687                 }
688         }
689
690         # Generierte Reverse Zones in Files schreiben
691
692         foreach my $sn (sort {$a <=> $b} keys %REVERSECONF) {
693         #       print $REVERSECONF{$sn};
694                 my $file = "db.134.109.$sn";
695                 rpc_buildstring("$ZONEBASE/$file", $REVERSECONF{$sn}, 'none');
696         }
697
698         # Serial aktualisieren
699
700         my $date = substr($serial, 0, 8);
701
702         $sql = qq{INSERT INTO dnsupdate (datum,revision) VALUES ('$date','$revision')};
703         $sth = db_call($dbh,$sql);
704         db_commit($dbh);
705
706         $dbh->disconnect();
707
708         return rpc_returnstring();
709         # Fertig ;-)
710 }
711
712 #
713 # Library functions
714 #
715
716 sub get_hostpart($) {
717         my ($a, $b, $c, $d) = split (/\./, shift);
718         return $d;
719 }
720
721
722 sub get_subnet($) {
723         my ($a, $b, $c, $d) = split (/\./, shift);
```

```
724         if ($c == 102 or $c == 103) {
725                 $c &= ~0x00;
726         } else {
727                 $c &= ~0x01;
728         }
729         return $c;
730 }
731
732
733 sub get_c_subnet($) {
734         my ($a, $b, $c, $d) = split (/\./, shift);
735         return $c;
736 }
737
738
739 sub rpc_buildstring($$$) {
740         my $file = shift;
741         my $data = shift;
742         my $change_perm = shift;
743
744       if ($change_perm eq 'none') {
745          push @filelist, RPC::XML::struct->new(
746            'file' => RPC::XML::string->new($file),
747            'data' => RPC::XML::base64->new($data)
748            );
749       } else {
750          push @filelist, RPC::XML::struct->new(
751            'file' => RPC::XML::string->new($file),
752            'data' => RPC::XML::base64->new($data),
753            'chown' => RPC::XML::string->new($change_perm)
754         );
755       }
756 }
757
758 sub replace_file($$) {
759         my $file = shift;
760         my $data = shift;
761
762         my $tmpl_file = "$file.tmpl";
763         my $out_file  = "$file.new";
764         open OUT, ">$out_file" or die "Cannot open $out_file for writing: $!";
765         if (-f $tmpl_file) {
766                 open TMPL, "<$tmpl_file" or die "Cannot open $tmpl_file for reading: $!";
767                 while (<TMPL>) {
768                         /^#RULES#/ or print OUT and next;
769                         print OUT $data;
770                 }
771                 close TMPL;
772         } else {
773                 print OUT $data;
774         }
775         close OUT;
```

```
776         rename "$file.new", "$file" or die "Cannot␣rename␣$file.new␣to␣$file:␣$!";
777 }
778
779 sub rpc_returnstring() {
780         my $data = RPC::XML::array->new(@filelist);
781         @filelist = ();
782         return $data;
783 }
784
785 sub ip_pack($) {
786         my $ip_str = shift;
787
788         my @ip_parts = split('\.', $ip_str, 4);
789         my $ip_num = 0;
790
791         while (@ip_parts) {
792                 my $ip_part = shift @ip_parts;  # shift: öhchstwertiges Byte zuerst
                        verarbeiten
793                 die "üungltiges␣IP-Format␣'$ip_str'" if ($ip_part < 0 or $ip_part > 255);
794                 $ip_num <<= 8;
795                 $ip_num += $ip_part
796         }
797
798         return $ip_num;
799 }
800
801 sub ip_unpack($) {
802         my $ip_num = shift;
803
804         my @ip_parts = ();
805         my $ip_str;
806
807         foreach (0..3) {
808                 unshift (@ip_parts, $ip_num & 255);
809                 $ip_num >>= 8;
810         }
811         die "gepackte␣IP-Adresse␣zu␣lang␣(älnger␣als␣32␣bit)" if $ip_num > 0;
812         $ip_str = join('.', @ip_parts);
813
814         return $ip_str;
815 }
816
817 sub subnet_enum_hosts($$) {
818         my $host    = ip_pack(shift);
819         my $netmask = ip_pack(shift);
820
821         my $subnet = $host & $netmask;
822         my $hostmask = ip_pack("255.255.255.255") - $netmask;
823         my @hosts = ();
824
825         for (my $ip = $subnet + 1; $ip < $subnet + $hostmask; $ip++) {
826                 my $ip_str = ip_unpack($ip);
```

```
827                     push @hosts, $ip_str;
828             }
829
830             return @hosts;
831     }
832
833     sub reformat_mac($) {
834             my $mac_hex6 = shift;
835             my @macs = split(':', $mac_hex6, 6);
836             if ($#macs != 5) {
837                     print "$mac_hex6 is not a valid mac address\n";
838                     return undef;
839             }
840             return "$macs[0]$macs[1].$macs[2]$macs[3].$macs[4]$macs[5]";
841     }
```

Listing 1: Sourcecode XML-RPC-Server

```
1   #!/usr/bin/perl -w -T
2   ################################################################################
3   #
4   # userconfig_get     (C) 2006 by
5   #                        Sebastian Wehrmann
6   #
7   # description:  holt nutzerbasierte Configfiles vom RPC-Server:
8   #               - firewall
9   #               - dynshaper
10  #               - static arp
11  #               - dhcpd
12  #               - tcp-resetter
13  #               - dns
14  #
15  # version:      $Id: userconfig_get 1000 2006-07-12 12:57:24Z sweh $
16  #
17  # license:      General Public License version 2
18  #
19  ################################################################################
20
21  use RPC::XML::Client;
22
23  sub fwconfig();
24  sub dhcpdconfig();
25  sub arpconfig();
26  sub dsconfig();
27  sub macconfig();
28  sub accountingconfig();
29  sub dnsconfig();
30
31  sub replace_file($);
32  sub log_error($);
33  sub handle_data($);
34
35  # Globale Defaults
```

```
36 ##############################################################################
37
38 my $RPCHOST = "localhost";
39 my $RPCPORT = "1663";
40
41 my $RPCCALLFW = "fwconfig";
42 my $RPCCALLDHCP = "dhcpdconfig";
43 my $RPCCALLARP = "arpconfig";
44 my $RPCCALLDS = "dsconfig";
45 my $RPCCALLMAC = "macconfig";
46 my $RPCCALLACC = "accountingconfig";
47 my $RPCCALLDNS = "dnsconfig";
48
49 # begin
50 ##############################################################################
51
52 my %oper;
53
54 for my $typ ("accounting","tcpreset","firewall","static_arp",
55              "dhcpd","dynshaper","dns","call_by_cron") {
56              $oper{$typ} = 0;
57 }
58
59 # TODO: Loeschen wenn umgestellt
60 if (scalar(@ARGV) == 0) {
61     $oper{call_by_cron} = 1;
62 #    exit(0);
63 }
64
65 while (my $param = shift) {
66         if ($param eq "-h" || $param eq "--help") {
67         print "$0: Neukonfiguration von Dateien\n";
68         print "--accounting\n";
69         print "--tcpreset\n";
70         print "--firewall\n";
71         print "--static_arp\n";
72         print "--dhcpd\n";
73         print "--dynshaper\n";
74         print "--dns\n";
75         print "--call_by_cron \t - alle Files\n";
76         exit(0);
77         }
78         my $paramkey = $param;
79         $paramkey =~ s/--//g;
80         if (not exists $oper{$paramkey}) {
81                 die "ungueltiges Argument '$param'\n".
82                 "Versuche '$0 -h' oder '$0 --help' fuer mehr Infos\n";
83         }
84         $oper{accounting} = 1 if $param eq "--accounting";
85         $oper{tcpreset} = 1 if $param eq "--tcpreset";
86         $oper{firewall} = 1 if $param eq "--firewall";
87         $oper{static_arp} = 1 if $param eq "--static_arp";
```

```
88          $oper{dhcpd} = 1 if $param eq "--dhcpd";
89          $oper{dynshaper} = 1 if $param eq "--dynshaper";
90          $oper{dns} = 1 if $param eq "--dns";
91          $oper{call_by_cron} = 1 if $param eq "--call_by_cron";
92  }
93
94  # Client init
95  my $rpccli = RPC::XML::Client->new('http://'.$RPCHOST.':'.$RPCPORT.'/RPCSERV');
96
97  if ($oper{call_by_cron} == 1) {
98      accountingconfig();
99      macconfig();
100     fwconfig();
101     dhcpdconfig();
102     arpconfig();
103     dnsconfig();
104     dsconfig();
105 } else {
106     accountingconfig() if $oper{accounting};
107     macconfig() if $oper{tcpreset};
108     fwconfig() if $oper{firewall};
109     arpconfig() if $oper{static_arp};
110     dhcpdconfig() if $oper{dhcpd};
111     dsconfig() if $oper{dynshaper};
112     dnsconfig() if $oper{dns};
113 }
114
115 exit(0);
116
117 # end
118 ##############################################################################
119 # begin subroutines
120
121 sub accountingconfig() {
122         my $rpcreq = $rpccli->send_request($RPCCALLACC);
123         handle_data($rpcreq);
124 }
125
126 sub macconfig() {
127         my $rpcreq = $rpccli->send_request($RPCCALLMAC);
128         handle_data($rpcreq);
129 }
130
131 sub fwconfig() {
132         my $rpcreq = $rpccli->send_request($RPCCALLFW);
133         handle_data($rpcreq);
134 }
135
136 sub dhcpdconfig() {
137         my $rpcreq = $rpccli->send_request($RPCCALLDHCP);
138         handle_data($rpcreq);
139 }
```

```
140
141 sub arpconfig() {
142         my $rpcreq = $rpccli->send_request($RPCCALLARP);
143         handle_data($rpcreq);
144 }
145
146 sub dsconfig() {
147         my $rpcreq = $rpccli->send_request($RPCCALLDS);
148         handle_data($rpcreq);
149 }
150
151 sub dnsconfig() {
152         my $rpcreq = $rpccli->send_request($RPCCALLDNS);
153         handle_data($rpcreq);
154 }
155
156 #
157 # Library functions
158 #
159
160 sub handle_data($) {
161   my $rpcreq = shift;
162
163   if ($rpcreq->is_fault) {
164     log_error($rpcreq);
165   } else {
166     foreach(@{$rpcreq->value}) {
167       replace_file($_);
168     }
169   }
170
171 }
172
173 sub replace_file($) {
174         my $databin = shift;
175         my $data = $databin->{data};
176         my $file = $databin->{file};
177         my $more = $databin->{chown};
178
179         my $tmpl_file = "$file.tmpl";
180         my $out_file  = "$file.new";
181         open OUT, ">$out_file" or die "Cannot␣open␣$out_file␣for␣writing:␣$!";
182         if (-f $tmpl_file) {
183                 open TMPL, "<$tmpl_file" or die "Cannot␣open␣$tmpl_file␣for␣reading:␣$!";
184                 while (<TMPL>) {
185                         /^#RULES#/ or print OUT and next;
186                         print OUT $data;
187                 }
188                 close TMPL;
189         } else {
190                 print OUT $data;
191         }
```

```
192           close OUT;
193           rename "$file.new", "$file" or die "Cannot␣rename␣$file.new␣to␣$file:␣$!";
194           if ($more) {
195             my ($login,$pass,$userid,$gid) = getpwnam($more)
196                   or die "User␣'$more'␣not␣in␣passwd␣file";
197             chown $userid, $gid, $file;
198           }
199   }
200
201
202   sub log_error($) {
203           my $fault = shift;
204
205           foreach my $key (keys %{$fault->value}) {
206             print ${$fault->value}{$key}."\n";
207           }
208   }
```

Listing 2: Sourcecode XML-RPC-Client